The Dissertation Committee for Ronald Lester Billings Certifies that this is the approved version of the following dissertation:

A Heuristic Method for Scheduling and Dispatching of Factory Production Using Multiclass Fluid Networks

Committee:

John J. Hasenbein, Supervisor

John W. Fowler

Erhan Kutanoğlu

Leon S. Lasdon

David P. Morton

A Heuristic Method for Scheduling and Dispatching of Factory Production Using Multiclass Fluid Networks

by

Ronald Lester Billings, B.S.E.E., M.B.A., M.S.E., M.S.C.A.M.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December, 2003

Dedication

This document is dedicated to my wife Judy, my daughter Kendra,
and my son Daniel in appreciation for the price they paid
in lack of attention for me to get this degree.
This document (and my life) is also dedicated to Jesus Christ
in appreciation for the price he paid in blood for me.

Acknowledgements

This material is based on work supported by the National Science Foundation under Grant No. 0128657. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the National Science Foundation.

A Heuristic Method for Scheduling and Dispatching of Factory Production Using Multiclass Fluid Networks

Publication No._____

Ronald Lester Billings, Ph.D.

The University of Texas at Austin, 2003

Supervisor: John J. Hasenbein

This dissertation describes a two-phase heuristic method for scheduling and dispatching production in a factory. In the first phase, the production flow is modeled as a multiclass fluid network. This fluid queueing model is a relaxation of the deterministic factory scheduling problem (in addition to being a limit of the stochastic queueing model) so it functions as an approximation of a discrete flexible job-shop with WIP and ongoing inputs. However, buffer levels are allowed to have non-integer values, equipment processing can be simultaneously shared between different products, and a single lot can begin processing at a downstream step before it completely finishes at the previous step. By solving a finite series of quadratic (or linear) programs, an optimal (or nearly optimal) control policy is found for this fluid relaxation problem (with a weighted holding cost objective).

In the second phase, production in the discrete factory queueing network is scheduled ahead of time or dispatched in real time by minimizing the deviation of the production from the optimal fluid control policy. Starting assignments are set with a mixed-integer program, and special techniques are used to deal with batching and to avoid sequence-dependent set-ups.

Table of Contents

List of Tables

List of Figures

List of Formulations

1. Introduction

1.1 OVERVIEW

Discrete manufacturing in general, and semiconductor factories (known as wafer fabs) in particular, have always presented very difficult scheduling and control problems. Features of wafer fabs that complicate analysis include reentrant product flows, large variation in processing times between different types of equipment (for example, 1 minute in a wet bench vs. 10 hours in a furnace), large variation in processing times at different stages on the same type of equipment, setup times for switching between product types, and batching at certain types of equipment. However, the cost of wafer fabs (over $3 billion for a new 300-mm factory) and the need to pay them off within three years means that efficient designs and operating strategies are absolutely essential. This dissertation evaluates the use of innovative multiclass fluid models for analyzing factory capacity and finding optimal (or nearly optimal) scheduling policies for semiconductor fabs.

Table 1 compares several methods for modeling the scheduling of factory production, from the complex (at the top of the table) to the simple (at the bottom). This scheduling problem is typically modeled as a deterministic job shop or multiclass queueing network. Neither formulation is of practical help in optimizing these systems exactly, and both formulations have conceptual failings. An approach with a finite horizon that considers only on-line scheduling policies unifies these two models, and can be approximated by a multiclass fluid network.

Table 1.1: Comparison of Factory Modeling Methods

Mathematical Model of the Factory	*Analysis Methods*			*Primary Conceptual Weakness*
	Capacity Evaluation	*Schedule Optimization*		
		Exact	*Heuristic*	
Job Shop: - discrete lots, steps, and equipment - deterministic (but can be averaged over a finite number of random instances in a stochastic program) - specific to individual lots	discrete-event (both resource- and job-driven) simulation: - well-defined - time-consuming to develop and run	combinatorial optimization: - NP-hard - only trivially sized problems can be solved exactly	many: - DOE or NLP with simulation - other ad hoc trial-and-error methods	- equal weight on current and future decisions - does not effectively handle ongoing input flows
Multiclass Queueing Network: - discrete lots, steps, and equipment - stochastic (often with memoryless inter-arrival and service distributions) - specific to each class (product type and process step)	classical queueing theory: - does not adequately represent reentrant product flows	Markov decision problem: - if memoryless inter-arrival and service distributions - only trivially sized problems can be solved exactly	diffusion approx-imation: - uses reflected Brownian motion in the space of machine work loads	- emphasis on stationary policies and steady state performance measures - does not effectively handle current work in progress

Mathematical Model of the Factory	*Analysis Methods*			*Primary Conceptual Weakness*
	Capacity Evaluation	*Schedule Optimization*		
		Exact	*Heuristic*	
On-Line Scheduling Policies (using only current queue lengths) with a Finite Horizon: - combination of both mathematical models (in above two rows)	same as above two rows	same as above two rows	multiclass fluid network (see row below)	- decisions made without knowledge of future data - emphasis on current work rather than steady state
Multiclass Fluid Network: - non-integer buffer levels - equipment shared between lots - processing of lots shared between steps	straight-forward arithmetic with direct implications for stability of discrete system	separated continuous linear program (SCLP) - solved by non-convex quadratic program or by simplex-like algorithm	linear program	- uses only means for inter-arrival and service times (and ignores distributions), which could be considered a strength

Figure 1.1 compares the trade-off in information value versus computational effort for several methods of evaluating factory capacity and performance. At the high end, to get a very accurate and complete picture of its capacity and performance, the actual physical factory could be operated for several months in real time. Job-driven and resource-driven simulations (such as those done in the AutoMod and Sigma simulation software environments, respectively) would respectively take days and hours to simulate the same

operating time with less and less fidelity. At the low end, static spreadsheet models (such as the SeMaTech Cost-Resource Model or CRM) run nearly instantaneously but give very poor results. Fluid models (such as those described in this document) fill in the gap between static spreadsheet models and simulation models.

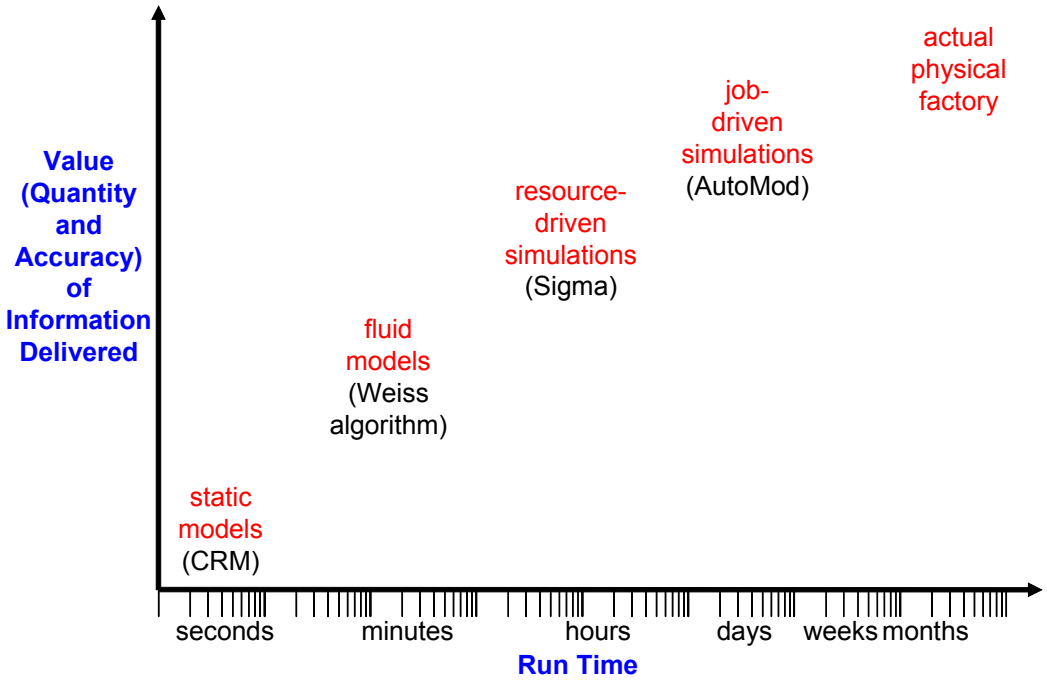

Figure 1.1: Methods for Evaluating Factory Capacity and Performance

The fluid control problem, which arises when modeling via multiclass fluid networks, is a relaxation of the wafer fab scheduling problem with on-line control policies, so it provides a bound on the optimal objective value. The fluid approach has been shown by Anderson [1981], Avram et al. [1995], Harrison [1996], and Weiss [1995] to have strong theoretical justification, and has recently

gained increasing importance as a scheduling and control tool (see Bertsimas and Gamarnik [1999], Bertsimas et al. [1999], Bertsimas and Sethuraman [2002], Dai and Weiss [2002], Maglaras [2000], Meyn [1997 and 2000]). The fluid control problem is a separated continuous linear program (SCLP) which is more tractable than Markov decision problems or combinatorial optimization problems. An SCLP with linear data can be solved numerically (as shown by Pullan [1993], Luo and Bertsimas [1998], and Fleischer and Sethuraman [2003]). It can also be solved using Weiss' [2002] finite simplex-based algorithm or by the quadratic and linear programming methods given in this document. These algorithms represent the state-of-the-art in methods for analyzing factory capacity and finding optimal (or nearly optimal) scheduling policies for semiconductor fabs.

A solution to the fluid control problem can be used to generate a detailed heuristic schedule for the wafer fab modeled by the fluid solution. A probabilistic analysis of the fluid control problem can also provide a probabilistic bound on the difference between the fluid solution and the heuristic objectives that bracket the optimum. For some cases, Dai and Weiss [2002] showed that the difference for a problem of size n is of order $\log(n)$.

1.2 JOB SHOPS

A job-shop formulation of a wafer fab is composed of individual machines (production equipment) and routes (product flows). Each route consists of ordered steps, each of which is processed on a specific type of machine with a given processing time. Jobs of each type arrive (or are released) into the factory at a known rate.

Producing a schedule for these jobs requires all of this data. The schedule determines start and completion times for each job step, and departure times. Typically in fab modeling, these times are deterministic (unlike equipment availability), but they can be averaged over a finite number of random instances in a stochastic program. Each of these times are constrained by order of steps, release times, processing times, and one-to-one assignment of jobs to machines. The objective function value of any schedule is some non-decreasing function of the departure times. There are a finite number of candidate schedules given by the sequence and a non-idling or non-delay policy (in which no machine is allowed to remain idle if it has work it can perform) is , so the optimal solution is well defined and (in theory) can be found by combinatorial optimization. However, such a problem is NP-hard, and no practical models of a size corresponding to real wafer fabs can be solved. Thus, heuristics must be used to generate a sub-optimal schedule.

Even optimal schedules (along with most heuristic schedules) are conceptually flawed, because they place equal weight on information about initial and later conditions. In a real manufacturing environment, the data for future decisions may be quite inaccurate. None of the models in this dissertation adequately address this problem.

1.3 MULTICLASS QUEUEING NETWORKS

A multiclass queueing network formulation does track specific jobs (individual product items). Instead, jobs are classified into groups (with common properties) from which individual jobs are arbitrarily chosen. Arrival times are typically assumed to come from a renewal process. Routing is more flexible: processing steps are listed by class, and a job that leaves one class moves to another class with a given probability. Processing times are assumed to be independent and identically distributed.

Problems to be solved include evaluating the performance of a set of given stationary policies (including measuring capacity or determining stability) and finding an optimal steady-state policy (which is a Markov decision problem on a discrete state-space for memoryless inter-arrival times and processing times, although elapsed times are sometimes included). Queueing theory fails to provide completely satisfactory tools for scheduling queueing networks with reentrant flows (such as found in wafer fabs). Figure 1.2 shows a generic semiconductor process flow using 130-nm copper interconnect and low-k dielectrics on 300-mm wafers. Even this ordering of tool types (the row headings) which minimizes the number of product flow reentry points has the flow go back to an earlier tool type 93 times. Column headings indicate different layers in the three-dimensional circuit design.

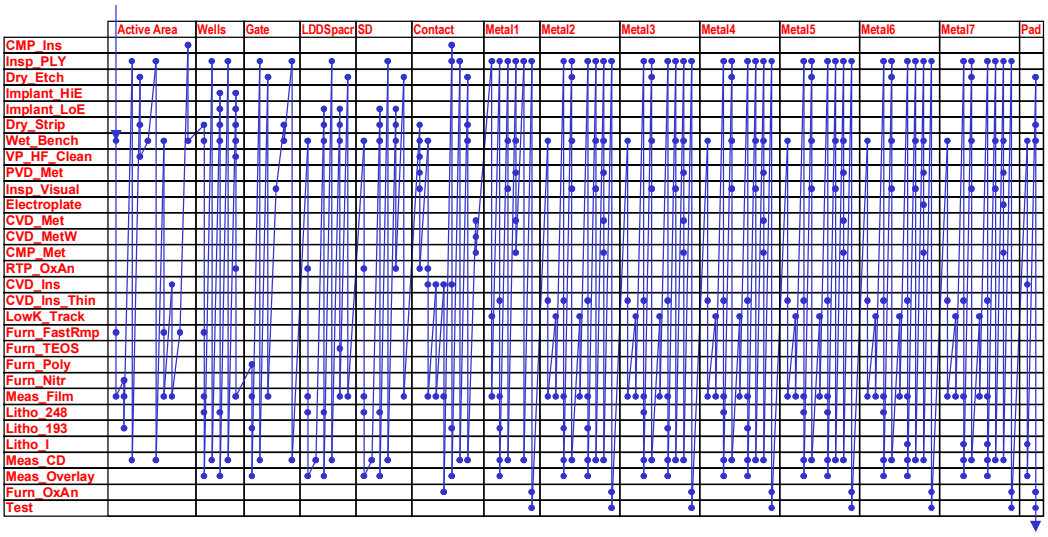

Figure 1.2: Generic Semiconductor Process Flow Showing Massively Reentrant Product Flows

A Markov decision problem can be solved in theory, but again no models of a size corresponding to real wafer fabs can be solved in practice. Heuristic approaches include diffusion approximations that assume reflected Brownian motion in the space of machine work loads (see Harrison [1988], Kushner and Ramachandran [1989], Maglaras [2000 and 2003], Meyn [2001], Veatch [2001 and 2002], and Wein [1992]). Again, the emphasis on stationary policies and steady state performance measures is conceptually flawed. In real wafer fabs, the initial system state is important, a solution is only needed for a finite time horizon, and the problem data is not reliable (or even stationary) beyond the near term.

1.4 ON-LINE POLICIES

A scheduling model that allows only on-line policies has features in common with both the deterministic model and the stochastic model (of the same real fab). For a decision at a specific time, schedulers restricted to an on-line policy use only the current queue length (or buffer content) in each class or buffer. Past history is not available to the scheduler. Additional information available to the scheduler includes only the average arrival rates, average service rates, and the routing. The time horizon is finite, and only the initial number of jobs in each queue and the following arrivals (over a finite horizon) will be scheduled. Only three objective functions are considered:

- We minimize the *makespan* (the time to completion of the last job).
- We minimize the integral over time of the *weighted holding cost* (the weighted buffer contents where the weights can be the WIP holding or inventory costs).
- We minimize the integral over time of the *maximum immediate workload* (the maximum over all of the machine types of the total time required for the machines of that type to process all of the jobs currently waiting for that machine type).

Such an on-line formulation erases the distinction between deterministic and stochastic models. The best on-line solution for a particular data instance can be compared to the (better) optimal combinatorial solution, as well as to any (worse) heuristic solution. An average of this comparison (over the population of instances indicated by the stochastic model) gives the expected performance of

the on-line formulation. For a probabilistic analysis of such methods (see Coffman and Lueker [1991]), one can compare the best on-line solution to the best combinatorial solution or to another on-line heuristic solution (such as a fluid approximation).

1.5 MULTICLASS FLUID NETWORKS

A multiclass fluid network is also composed of the same machines and buffers, but the buffer contents can take on any nonnegative real values. The system dynamics are deterministic but continuous. If all of the capacity of a machine is devoted to a specific buffer, then fluid leaves that buffer at a constant rate that is the inverse of the average processing time, and the fluid will then move on to the other buffers in proportion to the probabilities in the queueing model. The capacity of any machine is infinitely divisible, so fluid can leave each buffer at any rate between zero and the number of machines divided by the average processing time. Here the term "fluid" is used only in the non-technical sense of a continuous flow: the more complicated considerations of fluid dynamics (such as compressibility or viscosity) are not relevant. The general problem is to choose (as controls) pumping rates out of each buffer to minimize the objective function.

1.6 SCHEDULING HEURISTICS

A fluid heuristic for scheduling a flexible job shop is a two-phase process. First, using the data for the discrete job shop problem, a fluid relaxation problem is solved. Second, a feasible near-optimal schedule for the discrete job shop problem is generated from the optimal solution to the fluid relaxation. Figure 1.3 shows some algorithms that have been developed to perform these two tasks. See Appendix C to interpret the notation.

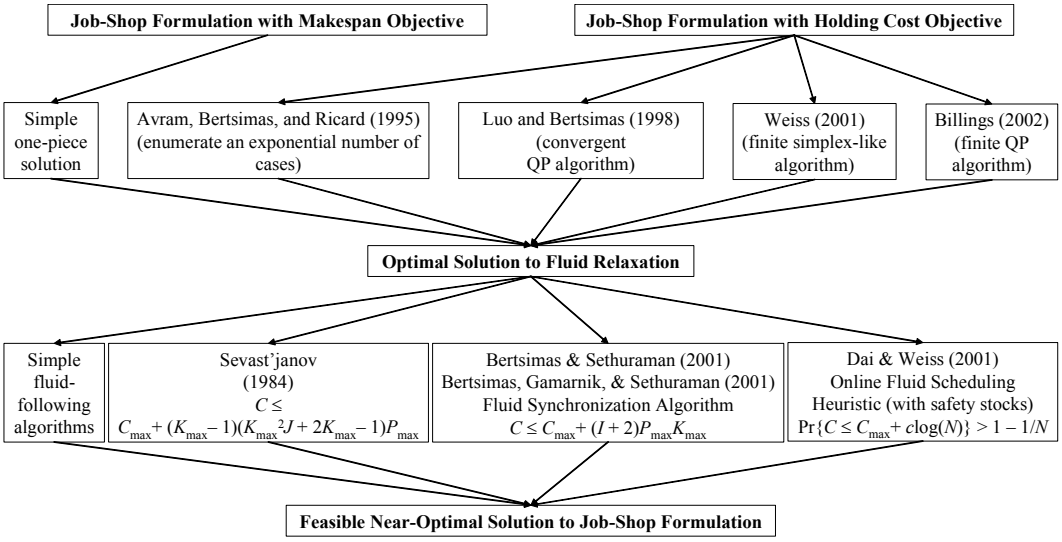

Figure 1.3: Two-Phase Fluid Heuristics for Scheduling a Flexible Job Shop

1.7 DISSERTATION OVERVIEW

Chapters 2 through 9 describe a variety of mathematical models of a factory. Chapter 2 discusses a deterministic job shop, while Chapter 3 analyses a stochastic job shop. Chapter 4 then presents a fluid continuous linear program (CLP) that can be adapted from either the deterministic or the stochastic job shop. Chapter 5 shows how the CLP model can be further transformed into a separated continuous linear program (SCLP) which has useful properties. Based on those properties, Chapter 6 demonstrates how the CLP is equivalent to a variety of quadratic programming (QP) models. Chapter 7 then reveals several linear programming (LP) heuristics for solving these models. Chapter 8 describes mixed integer programming (MIP) models for interpreting the solutions found in Chapters 5 through 7 for starting up a real factory or a discrete-event simulation (which is discussed in Chapter 9).

Chapter 10 gives the computational results for both optimization and simulation methods, and Chapter 11 wraps up the dissertation. Optimization code and simulation details are listed in Appendices A and B (respectively), and the notation is presented in Appendix C, just before the Bibliography.

2. Math Programming (MP) Models: Deterministic Discrete Job Shop

In a deterministic discrete flexible reentrant job shop (*FJc*|*recrc*|·), we have a set of *J* job types on *I* machine types (with M_i machines of each type *i*). Jobs (also called "lots") of type *j* are processed in K_j stages (also referred to as "tasks" or "classes"), each of which must be completed on a specified type of machine (with a "queue" or "buffer" holding the waiting jobs). The *k*th stage of the *j*th job type is represented by the pair (*j, k*) and has a processing time of length $p_{j,k}$ on machine type $\iota(j,k)$. We start at time t_0 with $q_{j,k}(t_0)$ jobs in each stage, and we have new jobs of type *j* arriving to their *k*th stage (although typically this only happens at the first stage) from outside the job shop at rate $\alpha_{j,k}$ (so the time between successive arrivals is $1/\alpha_{j,k}$).

The following decision variables (that depend on the scheduling or dispatching rules used) describe the trajectory of the system over time:

$q_{j,k}(t)$ the number of jobs of type *j* in queue or in service at stage *k* at time *t*,

$u_{j,k,m}(t)$ the cumulative number of units of job type *j* that have finished processing on specific machine *m* of type $\iota(j,k)$ at stage *k* by time *t*, and

$\tau_{j,k,m}(t)$ the cumulative equivalent amount of service time that machine *m* has spent processing jobs of type *j* at stage *k* by time *t*.

We want to schedule all the jobs through each of their tasks (in the prescribed sequence) subject to the following constraints:

1. The schedule is non-preemptive, so once a machine begins processing a task, it must complete that task before doing anything else.

2. Each machine can work on at most one task at a time.

3. A job can begin processing in each stage only after completing the previous stage.

The general formulation of the problem is to minimize an integral over a finite time horizon of length T subject to several constraints. This is an extension of the formulation given by Bertsimas and Sethuraman [2002] which is itself an extension of the classical job shop problem which is known to be NP-Hard.

The makespan objective function

$$C_{\max}\left(\mathbf{q},\mathbf{u},\boldsymbol{\tau}\,|\,T,...\right)=\int_{t_0}^{t_0+T}\left\langle\sum_{j=1}^{J}\sum_{k=1}^{K_j}q_{j,k}(t)>0\right\rangle dt \tag{2.1}$$

measures the time to complete all of the jobs that arrive by time t_0+T. The use of ellipses "..." in the objective function arguments is intended to indicate the existence of other data parameters besides T (such as t_0, J, K_j, etc.); they will only be explicitly written when needed for comparison. Here, $\langle\cdot\rangle$ is the indicator function which has a value of one if its logical argument is true and zero otherwise. The makespan objective makes little sense with ongoing arrivals, but it can be thought of as a surrogate objective for maximizing throughput if $\alpha_{j,k}=0$ for all j and k.

The weighted holding cost objective function

$$C_{\mathrm{h}}(\mathbf{q},\mathbf{u},\boldsymbol{\tau}\,|\,T,...)=\int_{t_0}^{t_0+T}\sum_{j=1}^{J}\sum_{k=1}^{K_j}c_{j,k}q_{j,k}(t)dt \tag{2.2}$$

measures the inventory cost over the time horizon. If the holding cost $c_{j,k}$ is one for all j and k, the weighted holding cost objective function divided by T measures

the average work in process. In that case, minimizing the holding cost objective function is equivalent to minimizing the average cycle time through the job shop (by Little's Law). If the time horizon is infinite, for Little's Law to hold, the system needs to have "regular departures" as defined by Serfozo [1999].

One problem with minimizing the weighted holding cost objective function is that it tends to starve different machine types unequally. On the other hand, the maximum workload objective function

$$C_{\mathrm{w}}(\mathbf{q},\mathbf{u},\boldsymbol{\tau}\,|\,T,...) = \int_{t_0}^{t_0+T} \max_i \left\{ \frac{1}{M_i} \sum_{(j,k)\in\sigma_i} p_{j,k} q_{j,k}(t) \right\} dt \tag{2.3}$$

integrates the largest workload (the time to process all jobs currently waiting) of the most heavily loaded machine type at each instant in time. Here, σ_i is the set of job types and stages that are processed on machines of type *i*. Minimizing the maximum workload objective function tends to feed different machine types equally. A weighted sum of the weighted holding cost objective function and the maximum workload objective function can also be a useful objective function.

The feasible domain of these objective functions is determined by the following constraints. First, flow must be conserved at the initial step for each job type,

$$\begin{aligned} \sum_{m=1}^{M_{\iota(j,1)}} \left[u_{j,1,m}(t) - u_{j,1,m}(t_0) \right] + q_{j,1}(t) \\ = q_{j,1}(t_0) + \left\lfloor (t-t_0)\alpha_{j,1} \right\rfloor \end{aligned} \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \end{cases}, \tag{2.4}$$

as well as at succeeding steps,

$$\begin{aligned}&\sum_{m=1}^{M_{\iota(j,k)}}\left[u_{j,k,m}(t)-u_{j,k,m}(t_0)\right]\\&-\sum_{m=1}^{M_{\iota(j,k-1)}}\left[u_{j,k-1,m}(t)-u_{j,k-1,m}(t_0)\right]\\&+q_{j,k}(t)=q_{j,k}(t_0)+\left\lfloor\left(t-t_0\right)\alpha_{j,k}\right\rfloor\end{aligned}\qquad\begin{cases}\forall t\geq t_0\\\forall j\in\{1,2,...,J\}\\\forall k\in\{2,3,...,K_j\}\end{cases},\qquad(2.5)$$

where the floor function "$\lfloor\cdot\rfloor$" returns the greatest integer less than or equal to its real-valued argument.

In the discrete job shop, only a whole number of jobs can wait for processing,

$$q_{j,k}(t)\in\mathbb{Z}_+\qquad\begin{cases}\forall t\geq t_0\\\forall j\in\{1,2,...,J\}\\\forall k\in\{1,2,...,K_j\}\end{cases},\qquad(2.6)$$

or complete processing,

$$u_{j,k,m}(t)\in\mathbb{Z}_+\qquad\begin{cases}\forall t\geq t_0\\\forall j\in\{1,2,...,J\}\\\forall k\in\{1,2,...,K_j\}\\\forall m\in\{1,2,...,M_{\iota(j,k)}\}\end{cases}.\qquad(2.7)$$

The number of jobs processed at each step is limited by the amount of processing time expended,

$$\frac{\tau_{j,k,m}(t)}{p_{j,k}}-u_{j,k,m}(t)\geq 0\qquad\begin{cases}\forall t\geq t_0\\\forall j\in\{1,2,...,J\}\\\forall k\in\{1,2,...,K_j\}\\\forall m\in\{1,2,...,M_{\iota(j,k)}\}\end{cases}.\qquad(2.8)$$

Note that an implication of the last two constraints along with the objective functions is that

$$u_{j,k,m}(t) = \left\lfloor \frac{\tau_{j,k,m}(t)}{p_{j,k}} \right\rfloor \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \\ \forall m \in \{1,2,...,M_{\iota(j,k)}\} \end{cases}.$$

At most one job stage at a time can be processed on each machine,

$$\sum_{(j,k)\in\sigma_i} \left\langle \frac{\tau_{j,k,m}(t)}{p_{j,k}} - u_{j,k,m}(t) > 0 \right\rangle \leq 1 \qquad \begin{cases} \forall t \geq t_0 \\ \forall i \in \{1,2,...,I\} \\ \forall m \in \{1,2,...,M_i\} \end{cases}. \qquad (2.9)$$

Similarly, a job can be processed in only one stage at a time, so the next stage cannot begin processing until the current stage is finished,

$$\frac{1}{p_{j,k}} \sum_{m=1}^{M_{\iota(j,k)}} \tau_{j,k,m}(t) - \sum_{m=1}^{M_{\iota(j,k-1)}} u_{j,k-1,m}(t) \leq q_{j,k}(t_0) \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}. \qquad (2.10)$$

Finally, the amount of processing time expended must be non-decreasing,

$$\tau_{j,k,m}(t') - \tau_{j,k,m}(t) \geq 0 \qquad \begin{cases} \forall t,t' : t_0 \leq t < t' \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \\ \forall m \in \{1,2,...,M_{\iota(j,k)}\} \end{cases}, \qquad (2.11)$$

and is limited by the amount of time available,

$$\sum_{(j,k)\in\sigma_i} \left[\tau_{j,k,m}(t') - \tau_{j,k,m}(t) \right] \leq t' - t \qquad \begin{cases} \forall t,t' : t_0 \leq t < t' \\ \forall i \in \{1,2,...,I\} \\ \forall m \in \{1,2,...,M_i\} \end{cases}. \qquad (2.12)$$

Putting all these equations together gives Formulation 2.1:

Formulation 2.1: Math Program (MP) for Deterministic Discrete Job Shop

$$\min_{\mathbf{q},\mathbf{u},\boldsymbol{\tau}} C_{\max}\left(\mathbf{q},\mathbf{u},\boldsymbol{\tau}\,|\,T,...\right) = \int_{t_0}^{t_0+T} \left\langle \sum_{j=1}^{J}\sum_{k=1}^{K_j} q_{j,k}(t) > 0 \right\rangle dt \qquad \text{or}$$

$$\min_{\mathbf{q},\mathbf{u},\boldsymbol{\tau}} C_{\mathrm{h}}(\mathbf{q},\mathbf{u},\boldsymbol{\tau}\,|\,T,...) = \int_{t_0}^{t_0+T} \sum_{j=1}^{J}\sum_{k=1}^{K_j} c_{j,k} q_{j,k}(t) dt \qquad \text{or}$$

$$\min_{\mathbf{q},\mathbf{u},\boldsymbol{\tau}} C_{\mathrm{w}}(\mathbf{q},\mathbf{u},\boldsymbol{\tau}\,|\,T,...) = \int_{t_0}^{t_0+T} \max_{i} \left\{ \frac{1}{M_i} \sum_{(j,k)\in\sigma_i} p_{j,k} q_{j,k}(t) \right\} dt$$

s.t.

$$\sum_{m=1}^{M_{\iota(j,1)}} \left[u_{j,1,m}(t) - u_{j,1,m}(t_0) \right] + q_{j,1}(t) = q_{j,1}(t_0) + \left\lfloor \alpha_{j,1}\left(t - t_0\right) \right\rfloor \qquad \begin{cases} \forall t \ge t_0 \\ \forall j \in \{1,2,...,J\} \end{cases}$$

$$\left.\begin{aligned} &\sum_{m=1}^{M_{\iota(j,k)}} \left[u_{j,k,m}(t) - u_{j,k,m}(t_0) \right] - \sum_{m=1}^{M_{\iota(j,k-1)}} \left[u_{j,k-1,m}(t) - u_{j,k-1,m}(t_0) \right] \\ &\qquad + q_{j,k}(t) = q_{j,k}(t_0) + \left\lfloor \alpha_{j,k}\left(t - t_0\right) \right\rfloor \\ &\frac{1}{p_{j,k}} \sum_{m=1}^{M_{\iota(j,k)}} \tau_{j,k,m}(t) - \sum_{m=1}^{M_{\iota(j,k-1)}} u_{j,k-1,m}(t) \le q_{j,k}(t_0) \\ &q_{j,k}(t) \in \mathbb{Z}_+ \end{aligned}\right\} \qquad \begin{cases} \forall t \ge t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}$$

$$\tau_{j,k,m}(t') - \tau_{j,k,m}(t) \ge 0 \qquad \begin{cases} \forall t,t' : t_0 \le t < t' \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \\ \forall m \in \{1,2,...,M_{\iota(j,k)}\} \end{cases}$$

$$\sum_{(j,k)\in\sigma_i} \left[\tau_{j,k,m}(t') - \tau_{j,k,m}(t) \right] \le t' - t \qquad \begin{cases} \forall t,t' : t_0 \le t < t' \\ \forall i \in \{1,2,...,I\} \\ \forall m \in \{1,2,...,M_i\} \end{cases}$$

$$\sum_{(j,k)\in\sigma_i} \left\langle \frac{\tau_{j,k,m}(t)}{p_{j,k}} - u_{j,k,m}(t) > 0 \right\rangle \le 1 \qquad \begin{cases} \forall t \ge t_0 \\ \forall i \in \{1,2,...,I\} \\ \forall m \in \{1,2,...,M_i\} \end{cases}$$

$$\left.\begin{aligned} &\frac{\tau_{j,k,m}(t)}{p_{j,k}} - u_{j,k,m}(t) \ge 0 \\ &u_{j,k,m}(t) \in \mathbb{Z}_+ \end{aligned}\right\} \qquad \begin{cases} \forall t \ge t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \\ \forall m \in \{1,2,...,M_{\iota(j,k)}\} \end{cases}$$

We can represent the data for this formulation in matrix or vector form (where all vectors are defined to be column vectors). Let $\kappa(j,k)$ be the cumulative stage index for jobs of type *j* in stage *k*:

$$\kappa(j,k) \equiv k + \sum_{j'=1}^{j-1} K_{j'} .$$

Then we define $\mathbf{p}$ to be a vector of processing times where element number $\kappa(j,k)$ has the value $p_{j,k}$. Similarly, $\mathbf{c}$ is a vector of holding costs where element number $\kappa(j,k)$ has the value $c_{j,k}$, $\boldsymbol{\alpha}$ is a vector of arrival rates where element number $\kappa(j,k)$ has the value $\alpha_{j,k}$, and $\mathbf{q}(t_0)$ is a vector of initial inventory where element number $\kappa(j,k)$ has the value $q_{j,k}(t_0)$. Decision variable $\mathbf{q}(t)$ is defined similarly. It is also useful to define decision variable $\mathbf{u}(t)$ as the vector of number of jobs that have finished processing (and moved on) where element number $\kappa(j,k)$ is the cumulative number of units of job type *j* that have finished processing (on any machine) at stage *k* by time *t* (including all units that finished processing before time t_0).

We also define $\mathbf{m}$ to be a vector of machine quantities where element number *i* has the value M_i, and we define $\mathbf{B}$ to be a binary matrix (of zeros and ones) showing the assignment of machines to job stages where element

$$B_{i,k} \equiv \begin{cases} 1 & \text{if } \exists (j,k') \in \sigma_i : \kappa(j,k') = k \\ 0 & \text{otherwise} \end{cases} .$$

Finally we define $\mathbf{P}$ to be the job routing (from-to) matrix which in this *simple* case consists of only zeros with some ones on the first superdiagonal:

$$\mathbf{P} = \begin{cases} \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} & \text{if } J = 1 \\ \text{a block diagonal matrix of such blocks} & \text{if } J > 1 \end{cases}. \tag{2.13}$$

Here, element (k,k') of matrix $\mathbf{P}$ is one if a job leaving stage k will proceed to buffer k' and zero otherwise. In the following chapters, we also allow more general cases where $\mathbf{P}$ can have other nonzero elements and nonzero elements less than unity. However, the row sums are still inclusively bounded between zero and unity:

$$\mathbf{0}_{K,1} \le \mathbf{P}\mathbf{1}_K \le \mathbf{1}_K$$

where $\mathbf{1}_K$ is the vector of length K containing all ones and $\mathbf{0}_{K,1}$ is the matrix with K rows and 1 column containing all zeros.

We also assume that arrivals from upstream (from other queues and from outside the system) to any queue are finite, so

$$\boldsymbol{\alpha} + \mathbf{P}^{\mathrm{T}}\boldsymbol{\alpha} + \left(\mathbf{P}^{\mathrm{T}}\right)^2\boldsymbol{\alpha} + \left(\mathbf{P}^{\mathrm{T}}\right)^3\boldsymbol{\alpha} + \cdots = \sum_{n=0}^{\infty}\left(\mathbf{P}^{\mathrm{T}}\right)^n\boldsymbol{\alpha} < \infty$$

which implies that the eigenvalues of $\mathbf{P}$ lie inside the unit circle. In that case, we have geometric convergence,

$$\sum_{n=0}^{\infty}\left(\mathbf{P}^{\mathrm{T}}\right)^n = \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}.$$

where $\mathbf{I}_K$ is the *K*-by-*K* identity matrix (not to be confused with *I*, the number of machine types).

We therefore define the vector of arrival rates from upstream to the queues by:

$$\boldsymbol{\alpha}^+ \equiv \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \boldsymbol{\alpha}$$

where the superscript plus sign implies that a variable vector is pre-multiplied by $\left(\mathbf{I}_K - \mathbf{P}\right)$ or $\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}$. Similarly, we define the vectors of total (immediate and upstream) initial WIP:

$$\mathbf{q}^+(t_0) \equiv \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \mathbf{q}(t_0),$$

and total (immediate and upstream) ongoing WIP:

$$\mathbf{q}^+(t) \equiv \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \mathbf{q}(t),$$

and the vector of keeping costs (the difference in holding cost between the current queue and the following queue) per unit time:

$$\mathbf{c}^+ \equiv \left(\mathbf{I}_K - \mathbf{P}\right)\mathbf{c}.$$

In general, $\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)$ is easily inverted. In the simple case,

$$\mathbf{I}_K - \mathbf{P}^{\mathrm{T}} = \begin{cases} \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 1 \end{bmatrix} & \text{if } J = 1 \\ \text{a block diagonal matrix of such blocks} & \text{if } J > 1 \end{cases}, \tag{2.14}$$

for which the inverse is a zero-one matrix (specifically, a block-diagonal matrix with each block a lower triangular matrix in which all elements in the lower triangle are equal to unity):

$$\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} = \begin{cases} \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 1 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 1 & 0 & 0 \\ 1 & 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & 1 & 1 & \cdots & 1 & 1 & 1 \end{bmatrix} & \text{if } J = 1 \\ \text{a block diagonal matrix of such blocks} & \text{if } J > 1 \end{cases}. \tag{2.15}$$

If $\mathbf{P}$ is not simple but has the same pattern of nonzeros as in the simple case (2.14), then $\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}$ has the same pattern of nonzeros as shown above in (2.15).

If $\mathbf{u}(t_0)$ (the initial number of jobs that have finished processing at each stage) is not known, then the following feasible lower bound can be used:

$$\mathbf{u}(t_0) = \left(\mathbf{I}_K - \mathbf{P}\right)^{-1} \mathbf{P}\mathbf{q}(t_0).$$

2.1 EXAMPLE NETWORK: MP FORMULATION

As an example of a flexible job shop, the diagram in Figure 2.1 shows the process flow for a simple job shop (with only one machine in each of two tool sets and one process flow with three steps).

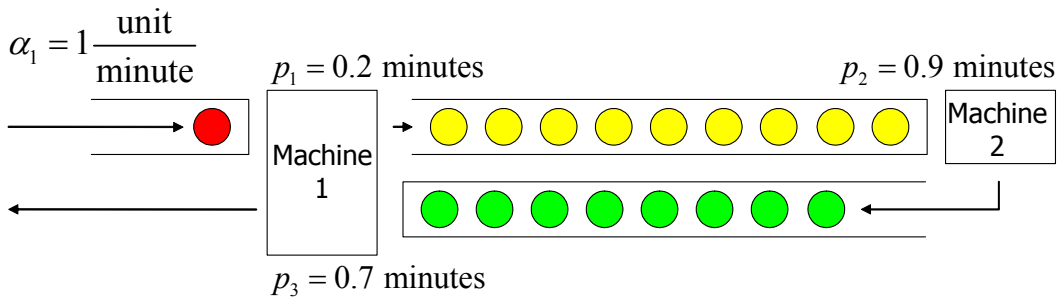

Figure 2.1: Example Process Flow

The system starts at time $t_0 = 0$, and the relevant vectors and matrices are the following:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \ \mathbf{p} = \begin{bmatrix} 0.2 \\ 0.9 \\ 0.7 \end{bmatrix}, \ \mathbf{c} = \mathbf{1}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \ \boldsymbol{\alpha} = \mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \ \mathbf{q}(t_0) = \begin{bmatrix} 1 \\ 9 \\ 8 \end{bmatrix}, \ \mathbf{u}(t_0) = \begin{bmatrix} 17 \\ 8 \\ 0 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \ \mathbf{m} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, . \mathbf{c}^+ = \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \ \boldsymbol{\alpha}^+ = \mathbf{1}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \text{ and } \mathbf{q}^+(t_0) = \begin{bmatrix} 1 \\ 10 \\ 18 \end{bmatrix}.$$

Figure 2.2 shows the total (immediate and upstream) WIP profile $\mathbf{q}^+(t)$ (which can be interpreted as the number of units in each buffer stacked on top each other) over time when a first-buffer-first-served (FBFS) policy is followed.

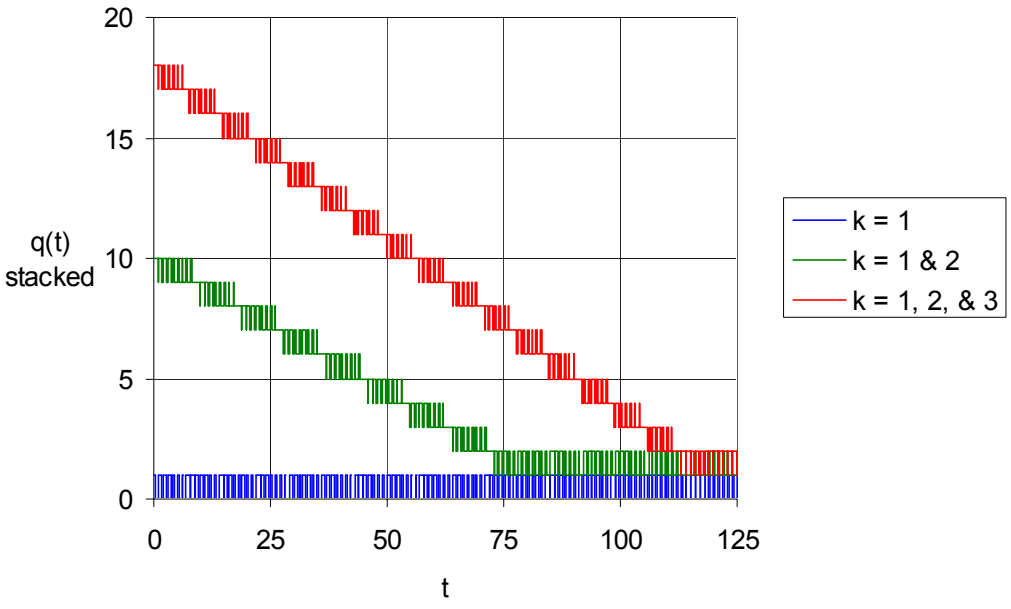

Figure 2.2: Total WIP Profile Under FBFS Policy for Example Network

Since

$$\mathbf{q}(t) \neq \mathbf{0} \qquad \forall t > 0$$

(so the buffers never completely drain), the makespan objective function is

$$C_{\max}\left(\mathbf{q}, \mathbf{u}, \boldsymbol{\tau} \mid T, \ldots\right) = T \qquad \forall T > 0 .$$

With these initial conditions, the system hits steady state around t = 113 (which is magnified in Figure 2.3).

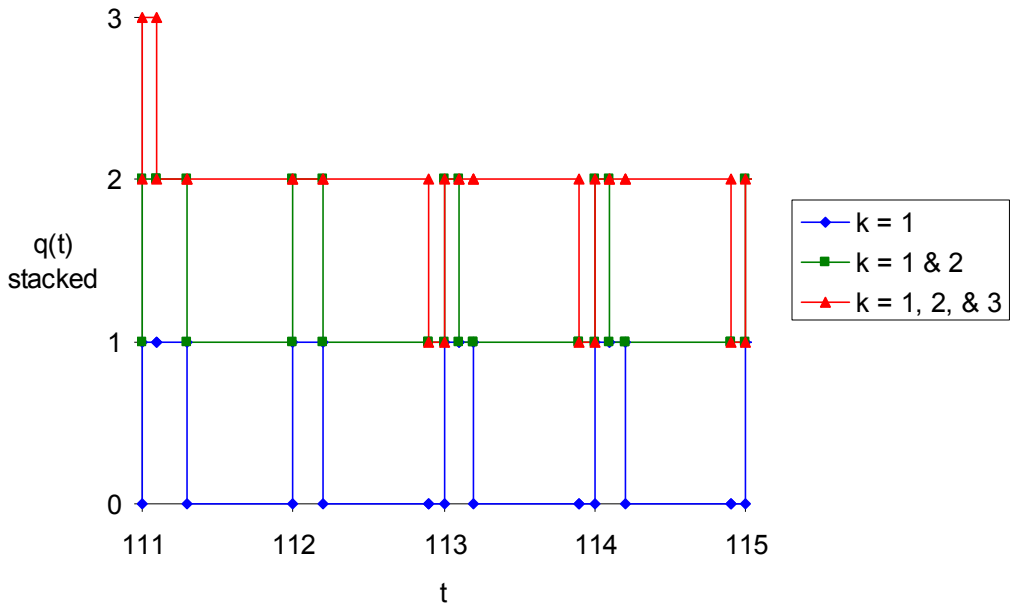

Figure 2.3: Example Network Reaches Steady State

The weighted holding cost objective function can be bounded by

$$C_{\mathrm{h}}(\mathbf{q},\mathbf{u},\boldsymbol{\tau}\,|\,T,\ldots) \in \begin{cases} \left[18-\dfrac{161}{2260}T\right]T \pm 1 & T \le 113 \\ \dfrac{18193}{20}+\dfrac{19}{10}T \pm 1 & T \ge 113 \end{cases}$$

so the long-run average total WIP level is given by

$$E\left[\|\mathbf{q}\|_1\right] \equiv \lim_{T\to\infty} \frac{C_{\mathrm{h}}(\mathbf{q},\mathbf{u},\boldsymbol{\tau}\,|\,T,\ldots)}{T}$$

$$= 1.9,$$

with the long-run average WIP levels for the individual buffers given by

$$E[\mathbf{q}] = \begin{bmatrix} 0.2 \\ 0.9 \\ 0.8 \end{bmatrix}.$$

These long-run average WIP levels appear to be independent of the initial inventory and the scheduling policy (as long as it does not allow idling or preemption).

2.2 EXTENSIONS

This model can be extended in several ways that are mentioned briefly in this section but the modeling and analysis of which are beyond the scope of this document. For example, capacity limits on WIP (for tactical or space-limitation reasons) may be represented by upper bounds on the elements of **q** (for each buffer) or **Bq** (for all the buffers at each machine type) or $\|\mathbf{q}\|_1$ (for the total WIP in the factory).

Process batch sizes can be comprehended by inserting a batch size term $\beta_{j,k}$ into the maximum workload objective function (2.3),

$$C_{\mathrm{w}}(\mathbf{q},\mathbf{u},\boldsymbol{\tau} \mid T,\ldots) = \int_{t_0}^{t_0+T} \max_i \left\{ \frac{1}{M_i} \sum_{(j,k)\in\sigma_i} \frac{p_{j,k} q_{j,k}(t)}{\beta_{j,k}} \right\} dt ,$$

and either by inserting $\beta_{j,k}$ in front of each instance of $u_{j,k,m}(t)$ in the formulation or by forcing the cumulative number of units that have finished processing to be a multiple of the batch size by changing constraint (2.7) to

$$u_{j,k,m}(t) \in \beta_{j,k}\mathbb{Z}^{+} \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \\ \forall m \in \{1,2,...,M_{\iota(j,k)}\} \end{cases}.$$

If partial batches are allowed, more complicated terms are required.

If sequence-dependent setup times $s_{j,k,j',k'}$ are required when switching processes, constraint (2.9) can be modified as follows:

$$\left\langle \frac{\tau_{j,k,m}(t)}{p_{j,k}} - u_{j,k,m}(t) > 0 \right\rangle + \left\langle \frac{\tau_{j',k',m}(t')}{p_{j',k'}} - u_{j',k',m}(t') > 0 \right\rangle \leq 1 \qquad \begin{cases} \forall t,t' : s_{j',k',j,k} \leq t' - t \leq s_{j,k,j',k'} \\ \forall i \in \{1,2,...,I\} \\ \forall j,k,j',k' : \begin{cases} (j,k) \in \sigma_i \\ (j',k') \in \sigma_i \\ (j,k) \neq (j',k') \end{cases} \\ \forall m \in \{1,2,...,M_i\} \end{cases}.$$

Machines that need to be taken out of service for scheduled preventive maintenance can have such down times modeled either as a single-stage job type with arrival rates and due dates or by not allowing processing to occur between times t_n and t'_n,

$$\sum_{(j,k)\in\sigma_i} \left[\tau_{j,k,m}(t'_n) - \tau_{j,k,m}(t_n) \right] = 0 .$$

The concept of "machines" can be expanded to cover any resource required at a given process step. For example operators (with specific skill sets) and fixtures (such as reticles for photo-lithography steps) may also be included in the set $\iota(j,k)$ of resources that are needed to process jobs of type *j* in stage *k*. However, if the model requires two or more resources to process any stage, it is no longer a simple multiclass queueing network.

Furthermore, if the transport of jobs from one machine to the next is also considered a process step, then the "machine" for that task is the material transporter which might be an automated material handling system (AMHS) vehicle or a human operator (possibly with a cart or forklift or other resource).

Due dates can be modeled as lower bounds on the cumulative number of units that have finished processing by each due date $d_{j,l}$,

$u_{j,K_j}(d_{j,l}) \geq l$.

Tight due dates may cause the problem to become infeasible, in which case a function of lateness or tardiness could become the objective function.

Also, if we set the holding cost to zero for the first buffer of each product, then the optimal solution (for the weighted holding cost objective function) suggests a start rate for each product.

Meyn [2001] suggested that a constant demand can be modeled as a final queue for which the external arrival rate is negative:

$\alpha_{j,K_j+1} < 0$.

3. Markovian Models: Stochastic Discrete Job Shop

In this chapter, we allow randomness in the job shop in four forms (and we will modify our example network from Chapter 2 to become stochastic as well). All probability distributions are assumed to be Markovian (or at least regenerative), although more general formulations are possible.

First, the time between successive arrivals to buffer (*j, k*) from outside the job shop is taken from a probability distribution on a non-negative domain (exponential for our example network) with mean $1/\alpha_{j,k}$. Second, the processing time at stage (*j, k*) is taken from a probability distribution on a non-negative domain (exponential for our example network) with mean $p_{j,k}$.

Third, as described in the previous chapter, the job routing (from-to) matrix **P** can have nonzero elements other than on the first superdiagonal and nonzero elements less than unity if product flows have scrap probabilities or sub-routes (such as rework loops or send-aheads or sample testing) or other probabilistic routings. Thus, element (k,k') of matrix **P** is the probability that a job leaving stage *k* will proceed to buffer k', and the sum of the elements of row *k* of matrix **P** is the probability that a job leaving stage *k* does not leave the system. For our example network, **P** will remain the same (so routing remains deterministic).

Finally, machines are allowed to randomly fail and be repaired, with the time until failure and time to repair taken from probability distributions on non-negative domains. Alternatively, the number or runs between failures can be taken from a positive discrete probability distribution. Machine reliability can

also be taken into account by defining a machine availability term a_i (the expected fraction of time that each machine of type *i* is available for processing). We then define $\mathbf{a}$ to be the vector of machine availabilities where element number *i* has the value a_i. For our example network (which was given in the previous chapter), we will assume no failures, so $\mathbf{a} = \mathbf{1}_2$.

It is useful to define the processing rate for jobs of type *j* in stage *k* as:

$$\mu_{j,k} \equiv \frac{1}{p_{j,k}}$$

and the traffic intensity at machine type *i* as

$$\rho_i \equiv \frac{\sum_{(j,k)\in\sigma_i} p_{j,k}\alpha^+_{j,k}}{a_i M_i},$$

with vector forms $\boldsymbol{\mu}$ and $\boldsymbol{\rho}$, respectively. For our example network, we have

$$\boldsymbol{\mu} = \begin{bmatrix} 5 \\ 10/9 \\ 10/7 \end{bmatrix}$$

and

$$\boldsymbol{\rho} = \begin{bmatrix} 0.9 \\ 0.9 \end{bmatrix}.$$

Throughout this document, we assume that

$$\rho_i < 1 \qquad \forall i \in \{1,2,...,I\},$$

so the arrivals do not overwhelm the processing capacity of the system (at least in the long term). In this chapter, we assume that all stochastic processes are stable

(positive recurrent), and we will focus on minimizing $E\left[\|\mathbf{q}\|_1\right]$ (the long-run average total WIP in the job shop) and on finding tight lower bounds on $E\left[\|\mathbf{q}\|_1\right]$ and $E[\mathbf{q}]$ (the long-run average individual queue levels).

3.1 LITTLE'S LAW

Little [1961] showed that, in the long run, the average WIP level is equal to the average throughput rate times the average cycle time through the system. If the system is positive recurrent, then the average throughput rate through each buffer (*j, k*) is the upstream arrival rate $\alpha^+_{j,k}$. The long-run average cycle time through each station (*j, k*) is bounded below by the mean processing time $p_{j,k}$. Thus, applying Little's Law yields:

$$E[\mathbf{q}] \geq \mathbf{D}(\mathbf{p})\boldsymbol{\alpha}^+$$

and

$$E\left[\|\mathbf{q}\|_1\right] \geq \mathbf{p}^{\mathrm{T}}\boldsymbol{\alpha}^+ .$$

Here $\mathbf{D}(\mathbf{x})$ is a diagonal matrix formed from any vector $\mathbf{x}$ (where diagonal element number *k* has the value x_k).

If we compute these values for our example network, we get

$$E[\mathbf{q}] \geq \begin{bmatrix} 0.2 \\ 0.9 \\ 0.7 \end{bmatrix}$$

and

$$E\left[\|\mathbf{q}\|_1\right] \geq 1.8$$

which is within 5.3% of the 1.9 value for $E\left[\|\mathbf{q}\|_1\right]$ in the deterministic job shop (a special case of the stochastic job shop).

3.2 POLYHEDRAL METHODS

Bertsimas et al. [1999] developed a better set of bounds on achievable performance for queueing networks that can be modeled by a continuous-time Markov chains (CTMC). For each of the $2^K - 1$ nonempty subsets $\mathbb{S} \subset \{1, 2, ..., K\}$, they defined a linear constraint on $\{x_k : k \in \mathbb{S}\}$ where x_k is the average cycle time through station *k*. This results in a linear program of the form

$$E\left[\|\mathbf{q}\|_1\right] \geq \min_{\mathbf{x}} \mathbf{x}^{\mathrm{T}} \boldsymbol{\alpha}^{+}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \geq \mathbf{b}$$

where constraint matrix **A** and vector **b** have $2^K - 1$ rows. Matrix **A** has K columns and at most $K2^{K-1}$ non-zero elements.

For our example network, matrix **A** and vector **b** have the form

$$\mathbf{A} = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 0.7 \\ 1.1 & 0.9 & 0 \\ 0.2 & 0 & 0.7 \\ 0 & 1.6 & 0.7 \\ 1.8 & 1.6 & 0.7 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 1/20 \\ 81/100 \\ 49/200 \\ 103/90 \\ 57/160 \\ 337/400 \\ 409/40 \end{bmatrix}$$

This results in the following bounds on individual queue lengths,

$$E[\mathbf{q}] \geq \mathbf{x} \geq \begin{bmatrix} 1/4 \\ 9/10 \\ 7/20 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.9 \\ 0.35 \end{bmatrix},$$

and total average WIP,

$$E\left[\|\mathbf{q}\|_1\right] \geq \left(\boldsymbol{\alpha}^+\right)^{\mathrm{T}} \mathbf{x}^* = \mathbf{1}_3^{\mathrm{T}} \begin{bmatrix} 427/90 \\ 9/10 \\ 7/20 \end{bmatrix} = \frac{1079}{180} = 5.99\overline{4}\,.$$

Note that the bound on the total average WIP is greater than the sum of the bounds on individual queue lengths.

3.3 ONE *M*/*GE*$_K$/*M* QUEUE

A general way of developing bounds on achievable performance is to analyze a simpler system; in essence, that is what we did in sections 3.1 and 3.2. Another simplification we can make is to allow any machine to perform any of the processing tasks, making the flexible job shop completely flexible. This simplification amounts to a relaxation of the constraints of the system, so the optimal scheduling policy for the simplified system must do at least as well as the optimal scheduling policy for the original system.

If $\mathbf{a} = \mathbf{1}_I$ so all machines are fully available, and if $\boldsymbol{\alpha} = \mathbf{e}_1$ so arrivals only show up at the first buffer, and if the job routing matrix **P** is "simple" which we define as having the form shown in equation (2.13) for the deterministic case, and if the time between successive arrivals to the first buffer and all processing times are exponentially distributed so the queueing network can be modeled by a CTMC, then an optimal scheduling policy is to have each machine take a job

from the first (arrival) queue and process that job through each of its K tasks until it leaves the system.

The result is a single $M/GE_K/M$ queue, where the first M indicates that the inter-arrival times come from a Markovian or memoryless (in other words, exponential) probability distribution, the GE_K indicates that the service times come from a generalized Erlang (also known as hypo-exponential) probability distribution (which consists of K phases, each of which is exponentially distributed), and the last M indicates that the total number of servers is

$$M \equiv \sum_{i=1}^{I} M_i \, .$$

The number of customers in such a queue follows a quasi birth-death process (QBDP) which has an infinitesimal generator matrix of the following form (where blank submatrices indicate zeros):

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{-2} & \mathbf{A}_{-1} & & & \\ \mathbf{A}_{-3} & \mathbf{A}_2 & \mathbf{A}_1 & & \\ & \mathbf{A}_3 & \mathbf{A}_2 & \mathbf{A}_1 & \\ & & \mathbf{A}_3 & \mathbf{A}_2 & \ddots \\ & & & \mathbf{A}_3 & \ddots \\ & & & & \ddots \end{bmatrix}.$$

The number of rows in $\mathbf{A}_{-2}$ and $\mathbf{A}_{-1}$ and the number of columns in $\mathbf{A}_{-2}$ and $\mathbf{A}_{-3}$ is given by

$$C(M+K-1,K) \equiv \binom{M+K-1}{K} \equiv \frac{(M+K-1)!}{K!(M-1)!}.$$

The number of rows in $\mathbf{A}_{-3}$, $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ and the number of columns in $\mathbf{A}_{-1}$, $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ is given by

$$C(M+K-1,M) \equiv \binom{M+K-1}{M} \equiv \frac{(M+K-1)!}{M!(K-1)!}.$$

The stationary probability vector $\mathbf{x}$ is the unique solution to the simultaneous linear equations

$$\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{x}$$

and

$$\mathbf{1}_{\infty}^{\mathrm{T}}\mathbf{x} = 1$$

Vector $\mathbf{x}$ gives the long-run fraction of time the QBDP spends in each state (where the state vector is $\mathbf{q}$).

Although $\mathbf{x}$ has infinite dimension, it can be computed using the matrix-geometric methods developed by Neuts [1981]. Let $\mathbf{x}_m$ be the subvector of $\mathbf{x}$ corresponding to the states for which $\|\mathbf{q}\|_1 = m$. Also let $\mathbf{R}$ be the unique solution matrix to the optimization problem

$$\min_{\mathbf{R}} |||\mathbf{R}|||$$

$$\text{s.t.} \quad \mathbf{R}^2\mathbf{A}_3 + \mathbf{R}\mathbf{A}_2 + \mathbf{A}_1 = \mathbf{0}_{N,N}$$

$$\mathbf{R} \in \mathbb{Z}_+^{C(M+K-1,M)\times C(M+K-1,M)}$$

where $|||\cdot|||$ is any matrix norm. Then we compute $\mathbf{x}_0$ through $\mathbf{x}_M$ as the unique solution vector to the simultaneous linear equations

$$\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_M \end{bmatrix}^{\mathbf{T}} \begin{bmatrix} \mathbf{A}_{-2} & \mathbf{A}_{-1} \\ \mathbf{A}_{-3} & \mathbf{A}_2 + \mathbf{R}\mathbf{A}_3 \end{bmatrix} = \mathbf{0}$$

and

$$\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{M-1} \end{bmatrix}^{\mathbf{T}} \mathbf{1}_{C(M+K-1,K)} + \mathbf{x}_M^{\mathbf{T}} \left(\mathbf{I} - \mathbf{R}\right)^{-1} \mathbf{1}_{C(M+K-1,M)} = 1 .$$

The remaining subvectors of **x** are given by

$$\mathbf{x}_n^{\mathbf{T}} = \mathbf{x}_M^{\mathbf{T}} \mathbf{R}^{n-M} .$$

This results in the following total average WIP for the single $M/GE_K/M$ queue (which is then a lower bound on the total average WIP for the original stochastic discrete job shop):

$$\begin{aligned} E\left[\left\|\mathbf{q}\right\|_1\right] &= \sum_{m=1}^{\infty} m\mathbf{x}_m^{\mathbf{T}}\mathbf{1} \\ &= \sum_{m=1}^{M-1} m\mathbf{x}_n^{\mathbf{T}}\mathbf{1} + \sum_{m=M}^{\infty} m\mathbf{x}_M^{\mathbf{T}}\mathbf{R}^{m-M}\mathbf{1} \\ &= \sum_{m=1}^{M-1} m\mathbf{x}_m^{\mathbf{T}}\mathbf{1} + \mathbf{x}_M^{\mathbf{T}}\left[\sum_{m=M}^{\infty} m\mathbf{R}^m\right]\mathbf{R}^{-M}\mathbf{1} \\ &= \sum_{m=1}^{M-1} m\mathbf{x}_m^{\mathbf{T}}\mathbf{1} + \mathbf{x}_M^{\mathbf{T}}\left[\left(\mathbf{I}-\mathbf{R}\right)^{-2}\mathbf{R} - \sum_{m=1}^{M-1} m\mathbf{R}^m\right]\mathbf{R}^{-M}\mathbf{1} \\ &= \sum_{m=1}^{M-1} m\left[\mathbf{x}_m^{\mathbf{T}}\mathbf{1} - \mathbf{x}_M^{\mathbf{T}}\mathbf{R}^{m-M}\mathbf{1}\right] + \mathbf{x}_M^{\mathbf{T}}\left(\mathbf{I}-\mathbf{R}\right)^{-2}\mathbf{R}^{1-M}\mathbf{1} . \end{aligned}$$

Our example network reduces to an $M/GE_3/2$ queue for which the state transition diagram has the form given in Figure 3.1:

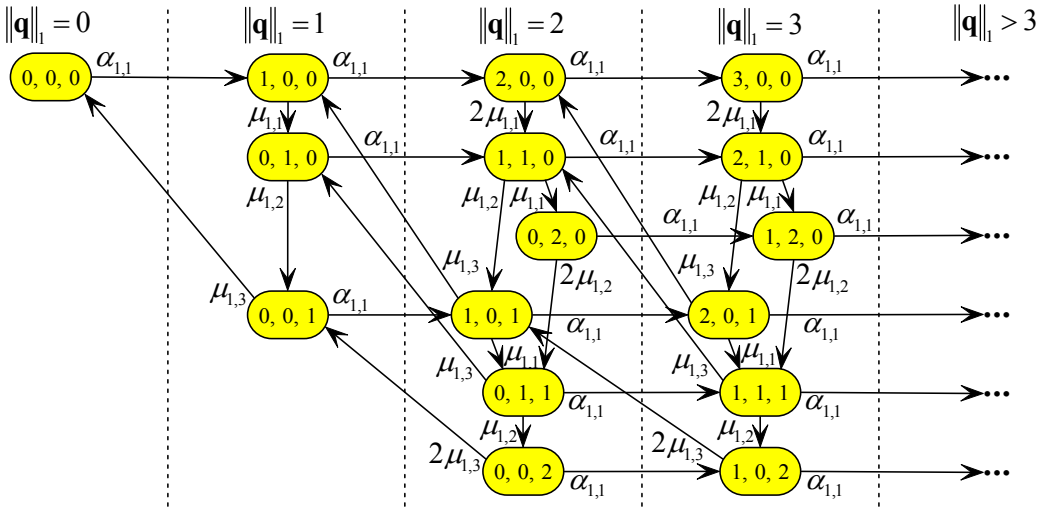

Figure 3.1: State Transition Diagram for $M/GE_3/2$ Queue From Example

The boundary submatrices of $\mathbf{A}$ have the following form (where lines separate subsubmatrices corresponding to different values of $\|\mathbf{q}\|_1 = m$ and states within each $\mathbf{x}_m$ sorted from top to bottom as on Figure 3.1):

$$\mathbf{A}_{-2} = \left[\begin{array}{c|ccc} 0 & \alpha_{1,1} & 0 & 0 \\ \hline 0 & -\mu_{1,1} & \mu_{1,1} & 0 \\ 0 & 0 & -\mu_{1,2} & \mu_{1,2} \\ \mu_{1,3} & 0 & 0 & -\mu_{1,3} \end{array}\right] - \alpha_{1,1}\mathbf{I}_4,\ \mathbf{A}_{-1} = \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \alpha_{1,1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha_{1,1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_{1,1} & 0 & 0 \end{array}\right], \text{ and}$$

$$\mathbf{A}_{-3} = \left[\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \mu_{1,3} & 0 & 0 \\ 0 & 0 & \mu_{1,3} & 0 \\ 0 & 0 & 0 & 2\mu_{1,3} \end{array}\right],$$

while the repeated submatrices of **A** have the following form:

$$\mathbf{A}_1 = \alpha_{1,1}\mathbf{I}_6,$$

$$\mathbf{A}_2 = \begin{bmatrix} -2\mu_{1,1} & 2\mu_{1,1} & 0 & 0 & 0 & 0 \\ 0 & -\mu_{1,1}-\mu_{1,2} & \mu_{1,1} & \mu_{1,2} & 0 & 0 \\ 0 & 0 & -2\mu_{1,2} & 0 & 2\mu_{1,2} & 0 \\ 0 & 0 & 0 & -\mu_{1,1}-\mu_{1,3} & \mu_{1,1} & 0 \\ 0 & 0 & 0 & 0 & -\mu_{1,2}-\mu_{1,3} & \mu_{1,2} \\ 0 & 0 & 0 & 0 & 0 & -2\mu_{1,3} \end{bmatrix} - \alpha_{1,1}\mathbf{I}_6, \text{ and}$$

$$\mathbf{A}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \mu_{1,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_{1,3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu_{1,3} & 0 & 0 \end{bmatrix}.$$

This results in the following total average WIP for the single $M/GE_3/2$ queue:

$$E\left[\|\mathbf{q}\|_1\right] \approx 7.24377.$$

The individual average queue lengths are then given by

$$E\left[q_{1,2}\right] = E\left[q_{1,2}\left|\left\|\mathbf{q}\right\|_1 = 1\right.\right]\mathbf{x}_1^{\mathbf{T}}\mathbf{1}_3 + E\left[q_{1,2}\left|\left\|\mathbf{q}\right\|_1 > 1\right.\right]\left(1 - \begin{bmatrix}\mathbf{x}_0\\ \mathbf{x}_1\end{bmatrix}^{\mathbf{T}}\mathbf{1}_4\right)$$

$$= \frac{p_{1,2}}{p_{1,1}+p_{1,2}+p_{1,1}}\mathbf{x}_1^{\mathbf{T}}\mathbf{1}_3 + 2\frac{p_{1,2}}{p_{1,1}+p_{1,2}+p_{1,1}}\left(1 - \begin{bmatrix}\mathbf{x}_0\\ \mathbf{x}_1\end{bmatrix}^{\mathbf{T}}\mathbf{1}_4\right)$$

$$= \frac{p_{1,2}}{p_{1,1}+p_{1,2}+p_{1,1}}\left(\mathbf{x}_1^{\mathbf{T}}\mathbf{1}_3 + 2 - 2\begin{bmatrix}\mathbf{x}_0\\ \mathbf{x}_1\end{bmatrix}^{\mathbf{T}}\mathbf{1}_4\right)$$

$$= \frac{p_{1,2}}{p_{1,1}+p_{1,2}+p_{1,1}}\left(2 - 2\mathbf{x}_0^{\mathbf{T}}\mathbf{1}_1 - \mathbf{x}_1^{\mathbf{T}}\mathbf{1}_3\right)$$

$$= 0.9$$

$$E\left[q_{1,3}\right] = \frac{p_{1,3}}{p_{1,1}+p_{1,2}+p_{1,1}}\left(2 - 2\mathbf{x}_0^{\mathbf{T}}\mathbf{1}_1 - \mathbf{x}_1^{\mathbf{T}}\mathbf{1}_3\right)$$

$$= 0.7$$

$$E\left[q_{1,1}\right] = E\left[\left\|\mathbf{q}\right\|_1\right] - E\left[q_{1,2}\right] - E\left[q_{1,3}\right]$$

$$\approx 5.64377$$

However, while the value of $E\left[\left\|\mathbf{q}\right\|_1\right]$ given here for the $M/GE_3/2$ queue does give a lower bound on the optimal long-run average total WIP in the example job shop, the values of $E\left[q_{1,1}\right]$, $E\left[q_{1,2}\right]$, and $E\left[q_{1,3}\right]$ given here do not give lower bounds on the long-run average individual queue levels in the example job shop.

3.4 *K* TANDEM *M*/*M*/*M*$_i$ QUEUES

Another simplification we can make is to duplicate machines so that each buffer (*j, k*) has $M_{\iota(j,k)}$ machines dedicated to it where $\iota(j,k)$ is the machine type that processes jobs of type *j* in stage *k*. Under the same assumptions as the previous section, the result is a set of *K* tandem *M*/*M*/*M*$_i$ queues, where the first two *M*s again indicate that the inter-arrival times and service times (respectively) come from a Markovian or memoryless probability distribution and the last *M* indicates that the number of servers at buffer (*j, k*) is $M_{\iota(j,k)}$. The traffic intensity at each dedicated queue (*j, k*) is then given by

$$\rho_{j,k} \equiv \frac{p_{j,k}\alpha^{+}_{j,k}}{M_{\iota(j,k)}}.$$

and the individual average queue lengths are given by:

$$E\left[q_{j,k}\right] = \frac{\alpha^{+}_{j,k}}{\mu_{j,k}} + \frac{\left(p_{j,k}\alpha^{+}_{j,k}\right)^{M_{\iota(j,k)}}\rho_{j,k}}{M_{\iota(j,k)}!\left(1-\rho_{j,k}\right)^{2}}\left\{\frac{\left(p_{j,k}\alpha^{+}_{j,k}\right)^{M_{\iota(j,k)}}}{M_{\iota(j,k)}!\left(1-\rho_{j,k}\right)} + \sum_{m=0}^{M_{\iota(j,k)}-1}\frac{\left(p_{j,k}\alpha^{+}_{j,k}\right)^{m}}{m!}\right\}^{-1}$$

which, if $M_{\iota(j,k)} = 1$, collapses to:

$$E\left[q_{j,k}\right] = \frac{\alpha^{+}_{j,k}}{\mu_{j,k} - \alpha^{+}_{j,k}}.$$

A coupling argument (such as given by Harrison and Wein [1989]) is needed to make this a rigorous lower bound.

If we compute these values for the three tandem *M*/*M*/1 queues suggested by our example network, we get

$$E[\mathbf{q}] = \begin{bmatrix} 0.25 \\ 9 \\ 2.\overline{3} \end{bmatrix}$$

and

$E\left[\|\mathbf{q}\|_1\right] = 11.58\overline{3}$.

Again, while the value of $E\left[\|\mathbf{q}\|_1\right]$ given here (for the three tandem $M/M/1$ queues) does give a lower bound on the optimal long-run average total WIP in the example job shop, the values of $E[\mathbf{q}]$ given here do not give lower bounds on the long-run average individual queue levels in the example job shop.

3.5 CONTINUOUS-TIME MARKOV DECISION PROCESS (CTMDP)

If all inter-arrival times and processing times and machine repair times are exponentially distributed, and machine failures are also memoryless (either the time to fail is exponentially distributed or the number or runs between failures is geometrically distributed), and if the machine scheduling/dispatching policies used are also memoryless (in the sense that no information is retained about the length of time individual lots have waited in the buffers), then the stochastic discrete job shop can be modeled perfectly by an infinite-dimensional continuous-time Markov decision process (CTMDP) which can also be seen as a stochastic dynamic program. However, CTMDP formulations are impractical for problems even slightly larger than our example network (and computationally infeasible for most full-size industrial scheduling problems). Also, for this example it can be shown that a stationary policy is optimal.

A simple CTMDP formulation for our example network (where we assume only non-preemptive machine scheduling/dispatching policies). has a state vector that consists of $\mathbf{q} \in \mathbb{Z}_+^3$ (the number of lots in each buffer), $k \in \{0,1,3\}$ (the buffer from which the first machine is currently processing, zero if idle), and $n \in \{0,1\}$ (one if the first machine began processing in this state, zero otherwise). Other constraints on feasible states include the following. First, a machine cannot process from an empty buffer:

$q_1 = 0 \Rightarrow k \neq 1$

and

$q_3 = 0 \Rightarrow k \neq 3$.

A machine cannot simultaneously begin processing and remain idle:

$k = 0 \Rightarrow n = 0$.

A non-idling policy is followed (which we assume for our example network, but can be deleted in other formulations):

$k = 0 \Rightarrow q_1 + q_3 = 0$.

If the first machine is not idle but did not begin processing in this state, then an arrival to the first buffer or a departure from the other machine must have occurred, so the first machine must have more than one lot in its buffers:

$n = 0 \neq k \Rightarrow q_1 + q_3 > 1$.

Finally, if the only lot in the system is in the third buffer (so the first machine is not idle), then the first machine must have initiated processing in this state (since

no arrival to the first buffer or departure from the other machine could have occurred):

$$n = 0 \Rightarrow \mathbf{q} \neq \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}}.$$

Thus, the set of feasible states is given by

$$\mathbb{S} = \left\{ \begin{bmatrix} \mathbf{q} \\ k \\ n \end{bmatrix} : \begin{array}{l} \mathbf{q} \in \mathbb{Z}_{+}^{3} \\ k \in \{0,1,3\} \\ n \in \{0,1\} \\ q_1 = 0 \Rightarrow k \neq 1 \\ q_3 = 0 \Rightarrow k \neq 3 \\ k = 0 \Rightarrow n = 0 \\ k = 0 \Rightarrow q_1 + q_3 = 0 \\ n = 0 \neq k \Rightarrow q_1 + q_3 > 1 \\ n = 0 \Rightarrow \mathbf{q} \neq \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} \end{array} \right\}.$$

Additional constraints on feasible states could include the following. If a first-buffer-first-served (FBFS) policy is followed (which constrains the problem to have only one feasible solution, so it takes the D out of CTMDP), then

$$\left(q_1 > 0\right) \wedge \left(q_3 > 0\right) \wedge \left(n = 1\right) \Rightarrow k = 1$$

where $\wedge$ is the binary logical And operator. If a last-buffer-first-served (LBFS) policy is followed (which similarly constrains the problem to have only one feasible solution), then

$$\left(q_1 > 0\right) \wedge \left(q_3 > 0\right) \wedge \left(n = 1\right) \Rightarrow k = 3.$$

The decision variables in these CTMDP formulations are the stationary probabilities $\{x_{\mathbf{s}} : \mathbf{s} \in \mathbb{S}\}$ (where x_s is the long-run fraction of time the CTMDP spends in each state $\mathbf{s} \in \mathbb{S}$). We fix

$$x_{\mathbf{s}} = 0 \qquad \forall \mathbf{s} \notin \mathbb{S}.$$

Our objective function is

$$\min_{\{x_{\mathbf{s}}:\mathbf{s}\in\mathbb{S}\}} E\left[\|\mathbf{q}\|_1\right] = \sum_{\mathbf{s}\in\mathbb{S}} x_{\mathbf{s}} \mathbf{s}^{\mathrm{T}} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

The law of total probability gives our first constraint on the decision variables:

$$\sum_{\mathbf{s}\in\mathbb{S}} x_{\mathbf{s}} = 1.$$

The remaining constraints on the decision variables are the local balance equations (which require that the rate at which the CTMDP departs from state **s** is the rate that it jumps from other states into **s**). The first of these constraints is for states in which the first machine is idle:

$$\begin{aligned} &\left(\alpha_{1,1} + \mu_{1,2}\langle q_2 > 0\rangle\right) x_{\begin{bmatrix} 0 & q_2 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}} \\ &= \mu_{1,1}\langle q_2 > 0\rangle \sum_{n\in\{0,1\}} x_{\begin{bmatrix} 1 & q_2-1 & 0 & 1 & n \end{bmatrix}^{\mathrm{T}}} \qquad \forall q_2 \geq 0. \\ &+ \mu_{1,3} \sum_{n\in\{0,1\}} x_{\begin{bmatrix} 0 & q_2 & 1 & 3 & n \end{bmatrix}^{\mathrm{T}}} \end{aligned}$$

Next, we have the constraints for states in which the first machine begins processing:

$$\sum_{k\in\{1,3\}}\left(\alpha_{1,1}+\mu_{1,k}+\mu_{1,2}\langle q_2>0\rangle\right)x_{[q_1\ \ q_2\ \ q_3\ \ k\ \ 1]^{\mathrm{T}}}$$
$$=\mu_{1,1}\langle q_2>0\rangle\sum_{n\in\{0,1\}}x_{[q_1+1\ \ q_2-1\ \ q_3\ \ 1\ \ n]^{\mathrm{T}}}$$
$$+\mu_{1,3}\sum_{n\in\{0,1\}}x_{[q_1\ \ q_2\ \ q_3+1\ \ 3\ \ n]^{\mathrm{T}}}$$
$$+\alpha_{1,1}\langle(q_1=1)\wedge(q_3=0)\rangle x_{[0\ \ q_2\ \ 0\ \ 0\ \ 0]^{\mathrm{T}}}$$
$$+\mu_{1,2}\langle(q_1=0)\wedge(q_3=1)\rangle x_{[0\ \ q_2+1\ \ 0\ \ 0\ \ 0]^{\mathrm{T}}}$$
$$\begin{cases}\forall q_1\geq 0\\ \forall q_2\geq 0\\ \forall q_3\geq 0\end{cases}.$$

Finally, we have the constraints for states in which the first machine neither begins processing nor is idle:

$$\left(\alpha_{1,1}+\mu_{1,k}+\mu_{1,2}\langle q_2>0\rangle\right)x_{[q_1\ \ q_2\ \ q_3\ \ k\ \ 0]^{\mathrm{T}}}$$
$$=\alpha_{1,1}\langle q_1>0\rangle\sum_{n\in\{0,1\}}x_{[q_1-1\ \ q_2\ \ q_3\ \ k\ \ n]^{\mathrm{T}}}$$
$$+\mu_{1,2}\langle q_3>0\rangle\sum_{n\in\{0,1\}}x_{[q_1\ \ q_2+1\ \ q_3-1\ \ k\ \ n]^{\mathrm{T}}}$$
$$\begin{cases}\forall q_1\geq 0\\ \forall q_2\geq 0\\ \forall q_3\geq 0\\ \forall k\in\{0,1\}\end{cases}.$$

Putting all these equations together gives Formulation 3.1:

Formulation 3.1: Infinite-Dimensional Linear Program (LP) for Continuous-Time Markov Decision Process (CTMDP) of Example Network

$$\min_{\{x_{\mathbf{s}}:\mathbf{s}\in\mathbb{S}\}} E\left[\left\|\mathbf{q}\right\|_1\right] = \sum_{\mathbf{s}\in\mathbb{S}} x_{\mathbf{s}} \mathbf{s}^{\mathrm{T}} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

s.t.

$$\sum_{\mathbf{s}\in\mathbb{S}} x_{\mathbf{s}} = 1$$

$$\left(\alpha_{1,1} + \mu_{1,2}\left\langle q_2 > 0\right\rangle\right) x_{\begin{bmatrix} 0 & q_2 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}}$$
$$= \mu_{1,1}\left\langle q_2 > 0\right\rangle \sum_{n\in\{0,1\}} x_{\begin{bmatrix} 1 & q_2-1 & 0 & 1 & n \end{bmatrix}^{\mathrm{T}}} + \mu_{1,3} \sum_{n\in\{0,1\}} x_{\begin{bmatrix} 0 & q_2 & 1 & 3 & n \end{bmatrix}^{\mathrm{T}}} \qquad \forall q_2 \geq 0$$

$$\sum_{k\in\{1,3\}} \left(\alpha_{1,1} + \mu_{1,k} + \mu_{1,2}\left\langle q_2 > 0\right\rangle\right) x_{\begin{bmatrix} q_1 & q_2 & q_3 & k & 1 \end{bmatrix}^{\mathrm{T}}}$$
$$= \mu_{1,1}\left\langle q_2 > 0\right\rangle \sum_{n\in\{0,1\}} x_{\begin{bmatrix} q_1+1 & q_2-1 & q_3 & 1 & n \end{bmatrix}^{\mathrm{T}}} + \mu_{1,3} \sum_{n\in\{0,1\}} x_{\begin{bmatrix} q_1 & q_2 & q_3+1 & 3 & n \end{bmatrix}^{\mathrm{T}}} \qquad \begin{cases} \forall q_1 \geq 0 \\ \forall q_2 \geq 0 \\ \forall q_3 \geq 0 \end{cases}$$
$$+ \alpha_{1,1}\left\langle \left(q_1 = 1\right) \wedge \left(q_3 = 0\right)\right\rangle x_{\begin{bmatrix} 0 & q_2 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}}$$
$$+ \mu_{1,2}\left\langle \left(q_1 = 0\right) \wedge \left(q_3 = 1\right)\right\rangle x_{\begin{bmatrix} 0 & q_2+1 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}}$$

$$\left(\alpha_{1,1} + \mu_{1,k} + \mu_{1,2}\left\langle q_2 > 0\right\rangle\right) x_{\begin{bmatrix} q_1 & q_2 & q_3 & k & 0 \end{bmatrix}^{\mathrm{T}}} \qquad \begin{cases} \forall q_1 \geq 0 \\ \forall q_2 \geq 0 \\ \forall q_3 \geq 0 \\ \forall k \in \{0,1\} \end{cases}$$
$$= \alpha_{1,1}\left\langle q_1 > 0\right\rangle \sum_{n\in\{0,1\}} x_{\begin{bmatrix} q_1-1 & q_2 & q_3 & k & n \end{bmatrix}^{\mathrm{T}}}$$
$$+ \mu_{1,2}\left\langle q_3 > 0\right\rangle \sum_{n\in\{0,1\}} x_{\begin{bmatrix} q_1 & q_2+1 & q_3-1 & k & n \end{bmatrix}^{\mathrm{T}}}$$

$$\mathbb{S} = \left\{ \begin{bmatrix} \mathbf{q} \\ k \\ n \end{bmatrix} : \begin{array}{l} \mathbf{q} \in \mathbb{Z}_+^3 \\ k \in \{0,1,3\} \\ n \in \{0,1\} \\ q_1 = 0 \Rightarrow k \neq 1 \\ q_3 = 0 \Rightarrow k \neq 3 \\ k = 0 \Rightarrow n = 0 \\ k = 0 \Rightarrow q_1 + q_3 = 0 \\ n = 0 \neq k \Rightarrow q_1 + q_3 > 1 \\ n = 0 \Rightarrow \mathbf{q} \neq \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} \end{array} \right\}$$

Of course, the problem with this formulation is that infinite-dimensional linear programs are not directly solvable. In order to truncate the problem, we introduce another constraint on the feasible states,

$$\|\mathbf{q}\|_1 \le q_{\text{Bound}},$$

where q_{Bound} is an upper limit on the total WIP in the job shop in the sense that arrivals cease while $\|\mathbf{q}\|_1 = q_{\text{Bound}}$. This triggers yet another constraint on the feasible states. If $q_1 + q_3 = q_{\text{Bound}} > 1$, then $q_2 = 0$, so the first machine could not have just finished processing a lot from the first buffer. Because $\|\mathbf{q}\|_1 = q_{\text{Bound}}$, it could not have just finished processing a lot from the third buffer. Since $q_1 + q_3 > 1$, it could not have been idle in the previous state. Thus,

$$(k \ne 0) \wedge (n = 1) \Rightarrow q_1 + q_3 < \max\{q_{\text{Bound}}, 2\}.$$

The set of feasible states is now given by

$$\mathbb{S} = \left\{ \begin{bmatrix} \mathbf{q} \\ k \\ n \end{bmatrix} : \begin{array}{l} \mathbf{q} \in \mathbb{Z}_+^3 \\ \|\mathbf{q}\|_1 \le q_{\text{Bound}} \\ k \in \{0,1,3\} \\ n \in \{0,1\} \\ q_1 = 0 \Rightarrow k \ne 1 \\ q_3 = 0 \Rightarrow k \ne 3 \\ k = 0 \Rightarrow n = 0 \\ k = 0 \Rightarrow q_1 + q_3 = 0 \\ n = 0 \ne k \Rightarrow q_1 + q_3 > 1 \\ n = 0 \Rightarrow \mathbf{q} \ne \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} \\ (k \ne 0) \wedge (n = 1) \Rightarrow q_1 + q_3 < \max\{q_{\text{Bound}}, 2\} \end{array} \right\}.$$

The constraints for states in which the first machine is idle are now

$$\begin{aligned}
&\left(\alpha_{1,1}\left\langle q_2 < q_{\text{Bound}}\right\rangle + \mu_{1,2}\left\langle q_2 > 0\right\rangle\right) x_{[0 \quad q_2 \quad 0 \quad 0 \quad 0]^{\mathrm{T}}} \\
&= \mu_{1,1}\left\langle q_2 > 0\right\rangle \sum_{n\in\{0,1\}} x_{[1 \quad q_2-1 \quad 0 \quad 1 \quad n]^{\mathrm{T}}} \qquad\qquad \forall q_2 : 0 \le q_2 \le q_{\text{Bound}}, \\
&+\mu_{1,3}\left\langle q_2 < q_{\text{Bound}}\right\rangle \sum_{n\in\{0,1\}} x_{[0 \quad q_2 \quad 1 \quad 3 \quad n]^{\mathrm{T}}}
\end{aligned}$$

the constraints for states in which the first machine begins processing are now

$$\begin{aligned}
&\sum_{k\in\{1,3\}} \left(\alpha_{1,1}\left\langle \|\mathbf{q}\|_1 < q_{\text{Bound}}\right\rangle + \mu_{1,k} + \mu_{1,2}\left\langle q_2 > 0\right\rangle\right) x_{[q_1 \quad q_2 \quad q_3 \quad k \quad 1]^{\mathrm{T}}} \\
&= \mu_{1,1}\left\langle q_2 > 0\right\rangle \sum_{n\in\{0,1\}} x_{[q_1+1 \quad q_2-1 \quad q_3 \quad 1 \quad n]^{\mathrm{T}}} \\
&+\mu_{1,3}\left\langle \|\mathbf{q}\|_1 < q_{\text{Bound}}\right\rangle \sum_{n\in\{0,1\}} x_{[q_1 \quad q_2 \quad q_3+1 \quad 3 \quad n]^{\mathrm{T}}} \qquad\qquad \forall \mathbf{q} \in \mathbb{Z}_+^3 : \|\mathbf{q}\|_1 \le q_{\text{Bound}}, \\
&+\alpha_{1,1}\left\langle (q_1 = 1) \wedge (q_3 = 0)\right\rangle x_{[0 \quad q_2 \quad 0 \quad 0 \quad 0]^{\mathrm{T}}} \\
&+\mu_{1,2}\left\langle (q_1 = 0) \wedge (q_3 = 1)\right\rangle x_{[0 \quad q_2+1 \quad 0 \quad 0 \quad 0]^{\mathrm{T}}}
\end{aligned}$$

and the constraints for states in which the first machine neither begins processing nor is idle are now

$$\begin{aligned}
&\left(\alpha_{1,1}\left\langle \|\mathbf{q}\|_1 < q_{\text{Bound}}\right\rangle + \mu_{1,k} + \mu_{1,2}\left\langle q_2 > 0\right\rangle\right) x_{[q_1 \quad q_2 \quad q_3 \quad k \quad 0]^{\mathrm{T}}} \\
&= \alpha_{1,1}\left\langle q_1 > 0\right\rangle \sum_{n\in\{0,1\}} x_{[q_1-1 \quad q_2 \quad q_3 \quad k \quad n]^{\mathrm{T}}} \qquad \begin{cases} \forall \mathbf{q} \in \mathbb{Z}_+^3 : \|\mathbf{q}\|_1 \le q_{\text{Bound}} \\ \forall k \in \{0,1\} \end{cases}. \\
&+\mu_{1,2}\left\langle q_3 > 0\right\rangle \sum_{n\in\{0,1\}} x_{[q_1 \quad q_2+1 \quad q_3-1 \quad k \quad n]^{\mathrm{T}}}
\end{aligned}$$

The state transition rate diagram for the finite-dimensional CTMDP has the form given in Figure 3.2 when $q_{\text{Bound}} = 1$ and in Figure 3.3 when $q_{\text{Bound}} = 2$:

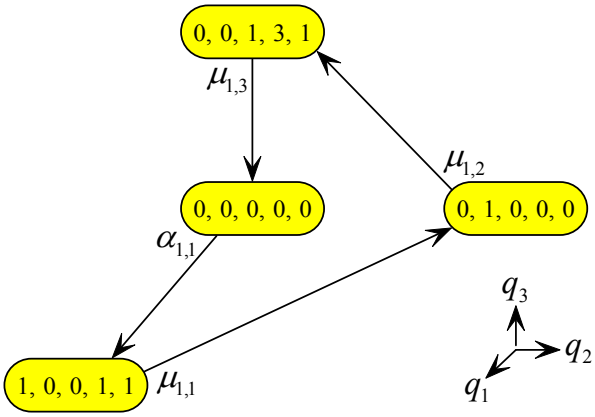

Figure 3.2: State Transition Rate Diagram for Finite-Dimensional CTMDP From Example When $q_{\text{Bound}} = 1$

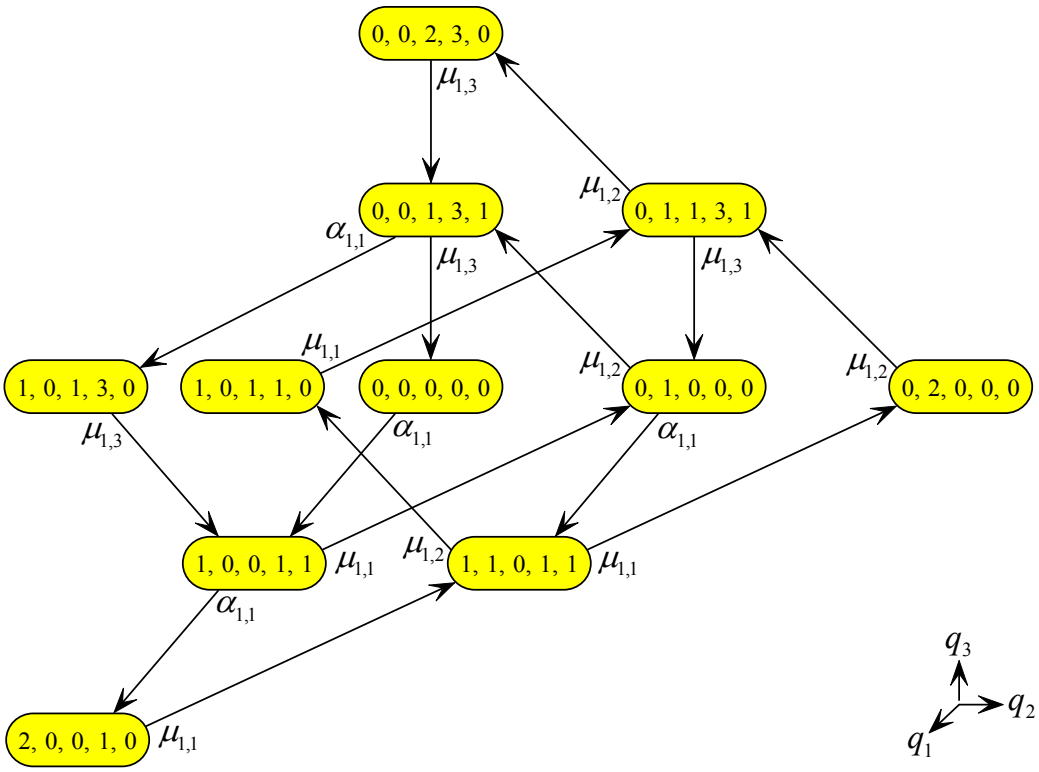

Figure 3.3: State Transition Rate Diagram for Finite-Dimensional CTMDP From Example When $q_{\text{Bound}} = 2$

When $q_{\text{Bound}} = 1$, we can see by inspection that

$$E[\mathbf{q}] = \begin{bmatrix} \dfrac{\dfrac{1}{\mu_{1,1}}}{\dfrac{1}{\alpha_{1,1}} + \dfrac{1}{\mu_{1,1}} + \dfrac{1}{\mu_{1,2}} + \dfrac{1}{\mu_{1,3}}} \\ \dfrac{\dfrac{1}{\mu_{1,2}}}{\dfrac{1}{\alpha_{1,1}} + \dfrac{1}{\mu_{1,1}} + \dfrac{1}{\mu_{1,2}} + \dfrac{1}{\mu_{1,3}}} \\ \dfrac{\dfrac{1}{\mu_{1,3}}}{\dfrac{1}{\alpha_{1,1}} + \dfrac{1}{\mu_{1,1}} + \dfrac{1}{\mu_{1,2}} + \dfrac{1}{\mu_{1,3}}} \end{bmatrix} = \begin{bmatrix} \frac{1}{14} \\ \frac{9}{28} \\ \frac{1}{4} \end{bmatrix} = \begin{bmatrix} 0.07\overline{142857} \\ 0.32\overline{142857} \\ 0.25 \end{bmatrix}$$

and

$$E\left[\|\mathbf{q}\|_1\right] = \frac{\dfrac{1}{\mu_{1,1}} + \dfrac{1}{\mu_{1,2}} + \dfrac{1}{\mu_{1,3}}}{\dfrac{1}{\alpha_{1,1}} + \dfrac{1}{\mu_{1,1}} + \dfrac{1}{\mu_{1,2}} + \dfrac{1}{\mu_{1,3}}} = \frac{9}{14} = 0.6\overline{428571}\,.$$

When $q_{\text{Bound}} = 2$, $E[\mathbf{q}]$ can be symbolically computed, but the result is far too complicated to be symbolically represented here.

Only when $q_{\text{Bound}} > 2$ does the first machine have a choice of which buffer to serve. If

$$\mathbf{q} \geq \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},$$

then both states $\begin{bmatrix}\mathbf{q}^{\mathrm{T}} & 1 & 1\end{bmatrix}^{\mathrm{T}}$ and $\begin{bmatrix}\mathbf{q}^{\mathrm{T}} & 2 & 1\end{bmatrix}^{\mathrm{T}}$ are feasible. Fortunately, any optimal solution for the LP formulation will give only one of the two states a nonzero probability of occurring, so

$$x_{\begin{bmatrix}\mathbf{q}^{\mathrm{T}} & 1 & 1\end{bmatrix}^{\mathrm{T}}} x_{\begin{bmatrix}\mathbf{q}^{\mathrm{T}} & 2 & 1\end{bmatrix}^{\mathrm{T}}} = 0 \qquad \forall \mathbf{q} \geq \begin{bmatrix}1\\0\\1\end{bmatrix}.$$

As q_{Bound} increases, the number of variables in the LP formulation (the number of feasible states) increases by $O\left(q_{\mathrm{Bound}}^3\right)$ and the number of non-zero elements in the coefficient matrix increases by $O\left(q_{\mathrm{Bound}}^6\right)$. Thus, the CTMDP rapidly becomes too large to solve. Figure 3.4 shows the optimal average WIP levels $E\left[\|\mathbf{q}\|_1\right]$ for small enough values of q_{Bound} that the LP formulation can be solved by GAMS/CPLEX 7. CPLEX actually crashed (probably for lack of memory) before finding an optimal solution for $q_{\mathrm{Bound}} > 52$, but because it was performing the dual simplex algorithm, the average WIP levels displayed form a lower bound on the optimal solutions for these and larger values of q_{Bound}. For larger values of q_{Bound} than those displayed here, CPLEX crashed even before finding a feasible solution. Also shown in Figure 3.4 are the average WIP levels that result from first-buffer-first-served (FBFS) policy, a last-buffer-first-served (LBFS) policy, and a tetrahedral policy (described in the following paragraphs).

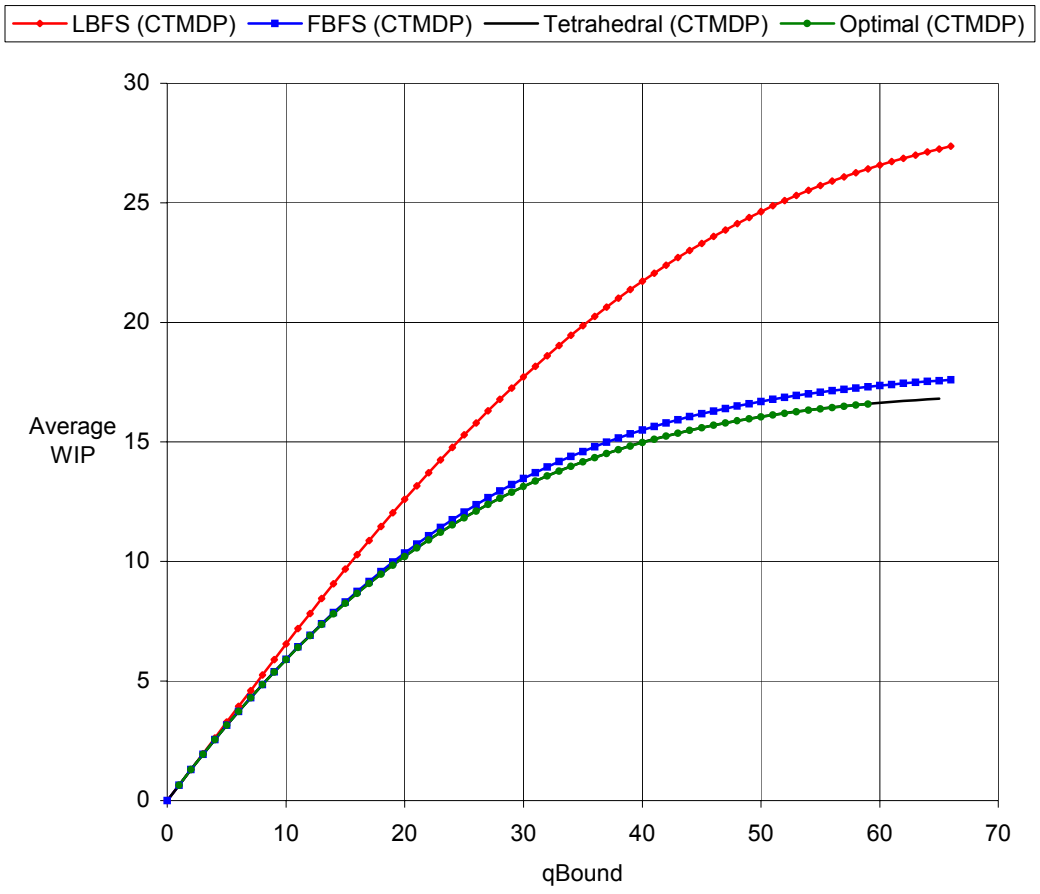

Figure 3.4: Average WIP Levels vs. q_{Bound}

For small enough values of q_{Bound} that the LP formulation can be solved, the solutions can give insight into the structure of solutions for larger values of q_{Bound}. For example, we may view the feasible values of **q** as a set of lattice points in $\mathbb{Z}_+^3$ and consider the boundary between the values of **q** for which the first machine serves the first buffer and the values of **q** for which the first machine serves the last buffer. In the optimal solution, this boundary is very closely approximated by a tetrahedron formed by four planes. Specifically, apart from

the obvious case when the first buffer is empty, the first machine serves the last buffer if and only if

$$\mathbf{q} \in \left\{ \mathbb{Z}_+^3 \;\middle|\; \begin{array}{l} q_1 \geq 0 \\ q_2 \geq 5.75\left[1-\exp\left(-0.05 q_{\text{Bound}}\right)\right] \\ \quad +0.17\exp\left(-0.02 q_{\text{Bound}}\right) q_1 \\ \quad -0.20\exp\left(-0.01 q_{\text{Bound}}\right) q_3 \\ q_2 \leq q_{\text{Bound}} - 2.5 \\ \quad -1.25 q_1 \\ \quad -1.01 q_3 \\ q_3 \geq 1 \end{array} \right\} \quad . \tag{3.1}$$

This tetrahedral policy was also plotted in Figure 3.4 and its relative error compared to the optimal solution is shown in Figure 3.5. Figures 3.6 through 3.11 show the values for the q_2 intercepts and the coefficients for q_1 and q_3 in equation (3.1) for the best tetrahedral approximation to the optimal solution for each value of q_{Bound}. Linear fits and an exponential fits for each of these three curves are also plotted to show why equation (3.1) has the form shown.

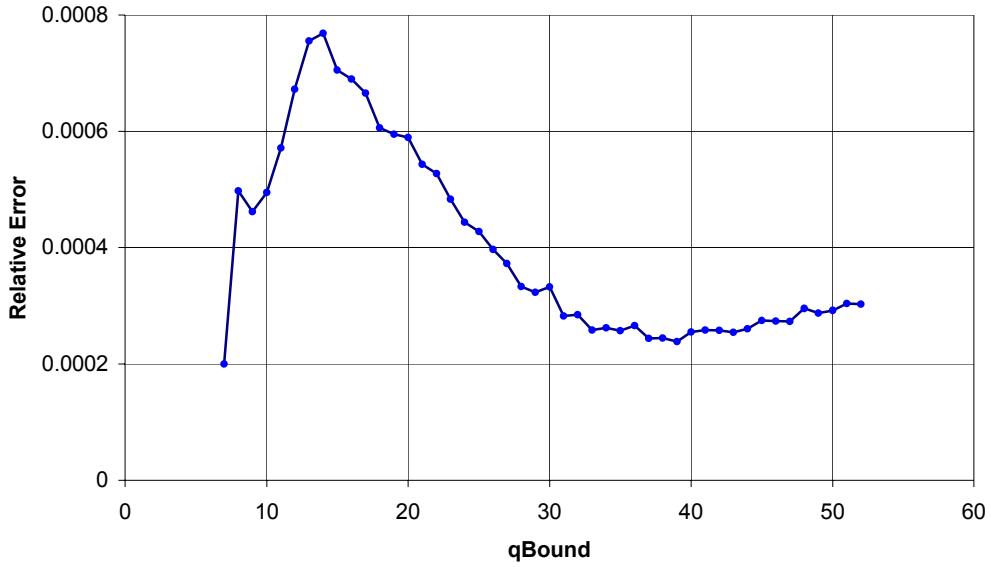

Figure 3.5: Relative Error of Tetrahedral Policy vs. q_{Bound}

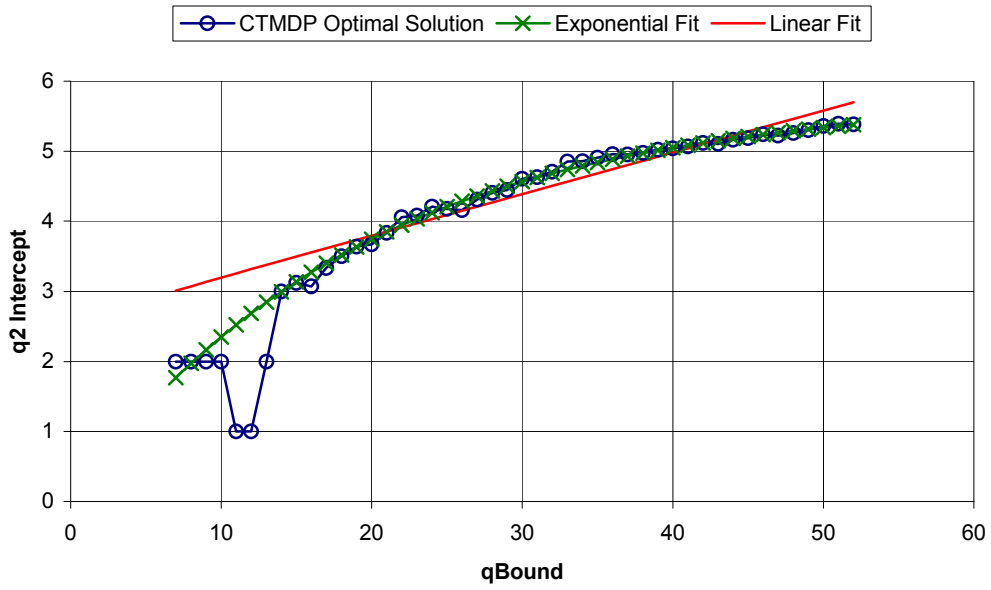

Figure 3.6: q_2 Intercepts for Lower Boundary Plane vs. q_{Bound}

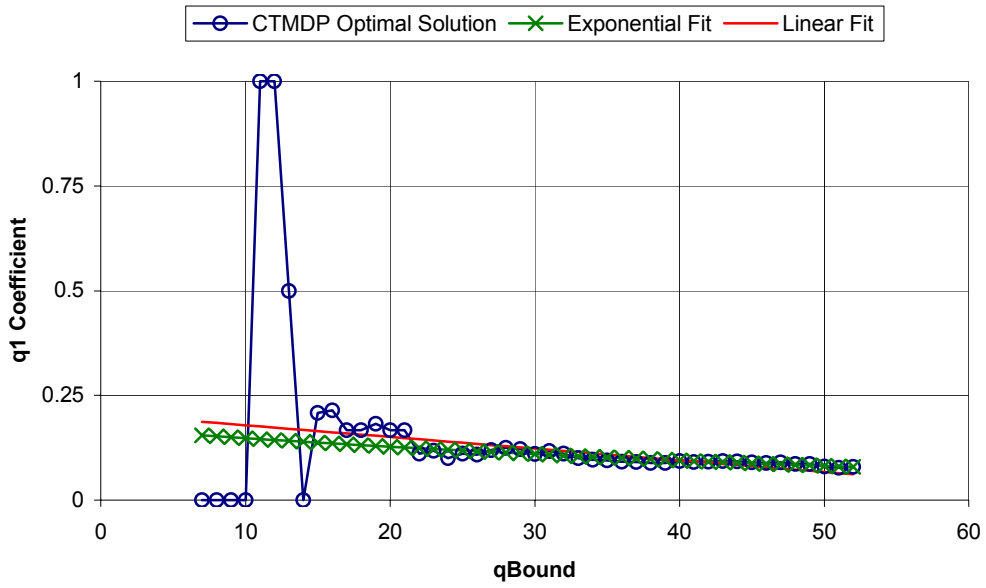

Figure 3.7: q_1 Coefficients for Lower Boundary Plane vs. q_{Bound}

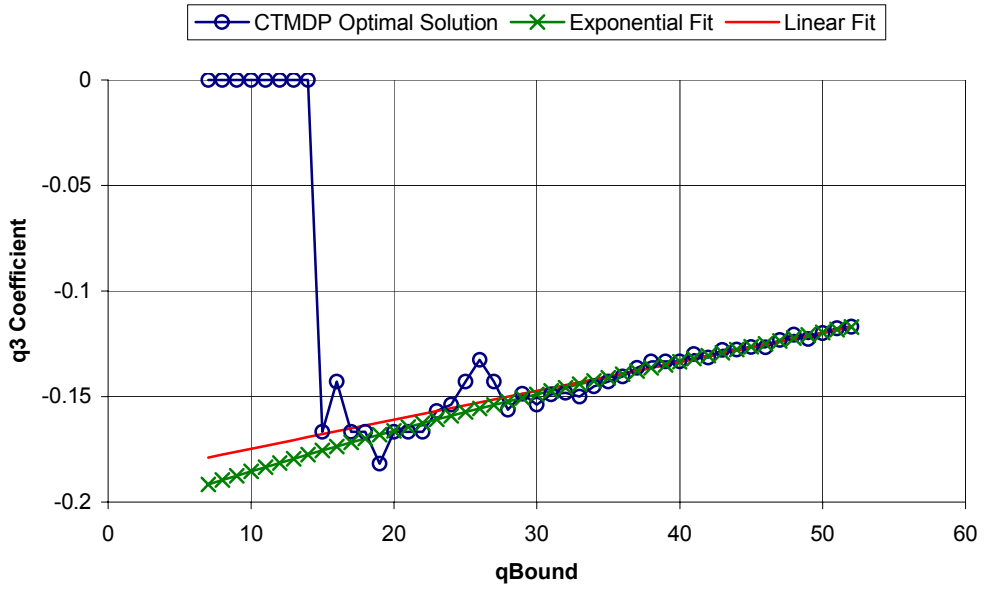

Figure 3.8: q_3 Coefficients for Lower Boundary Plane vs. q_{Bound}

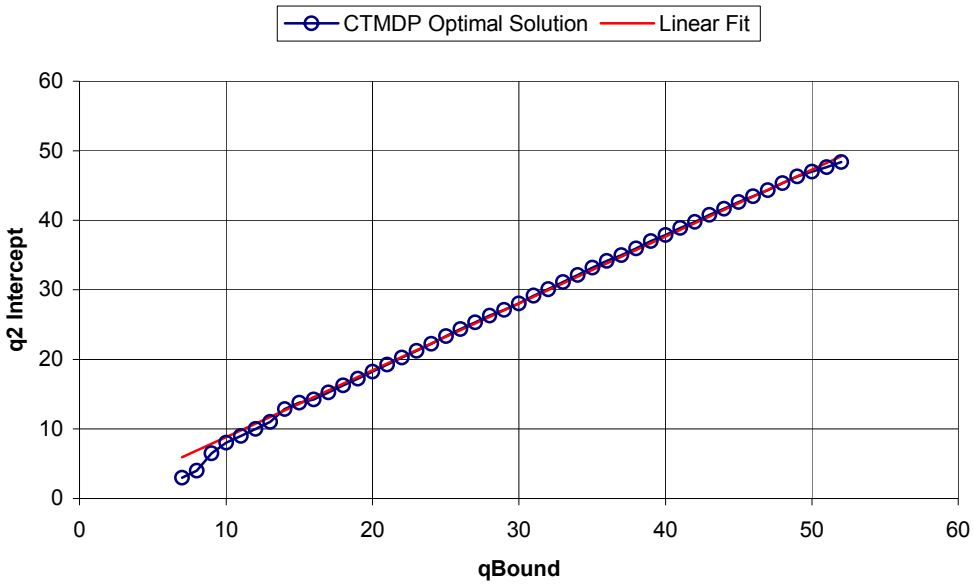

Figure 3.9: q_2 Intercepts for Upper Boundary Plane vs. q_{Bound}

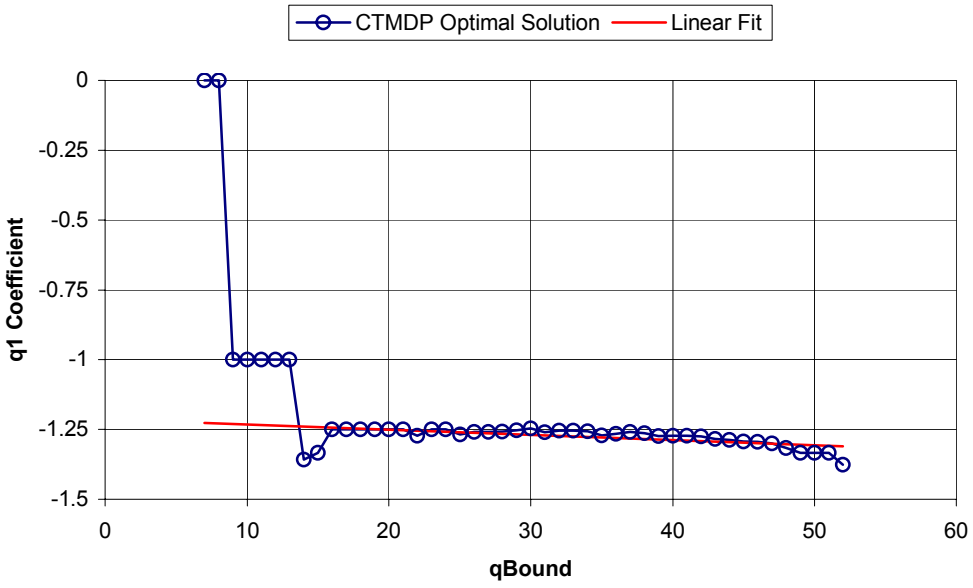

Figure 3.10: q_1 Coefficients for Upper Boundary Plane vs. q_{Bound}

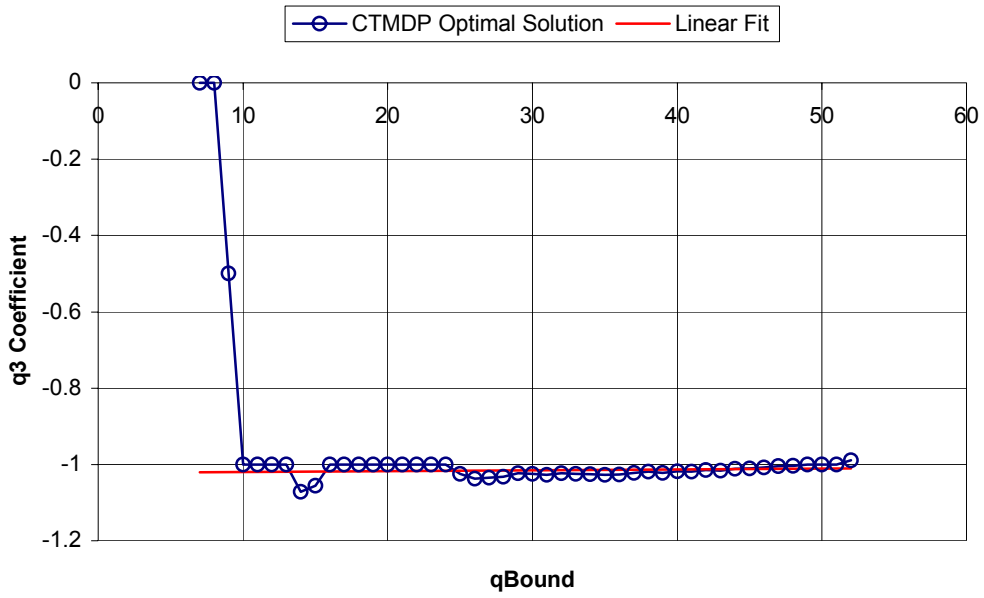

Figure 3.11: q_3 Coefficients for Upper Boundary Plane vs. q_{Bound}

The nearly tetrahedral form of the optimal solution has two important implications. First, although it is rather counter-intuitive, the optimal solution sometimes has the first machine serve the last buffer even when the second buffer is empty (so the second machine is starving). Second, as q_{Bound} increases, the optimal policy seems to be converging to a threshold policy similar to the policy proposed by Harrison and Wein [1989] and Martins et al. [1996] (for a 2-product, 2-machine, 3-buffer, non-reentrant criss-cross network). In such a threshold policy, the first machine serves the first buffer if and only if the last buffer is empty or the second buffer contains fewer than six items.

In addition to the LP formulation, a CTMDP can also be solved by other methods such as value iteration. It is also possible to develop a CTMDP

formulation for our example network that has a state vector that consists of only $\mathbf{q}$ and k (with or without the $k = 0$ states) such as was developed by Yang [1996]. However, the transition probabilities become rather cumbersome to derive, and the fraction of time at each WIP level would not be known. On the other hand, such a CTMDP formulation would have roughly half as many states (with a overwhelming increase in the density of non-zero elements in the coefficient matrix) and could be transformed via a uniformization process into a discrete-time Markov decision process (DTMDP) for much faster solution.

3.6 SUMMARY

Figure 3.12 shows the queue levels given by the different models described in this chapter, as well as a simulation model described in Chapters 9 and 10. Again, while the value of the average total WIP given here (for the different models) does give a lower bound on the optimal long-run average total WIP in the example job shop, the WIP values for the individual stages given here do not give lower bounds on the long-run average individual queue levels in the example job shop.

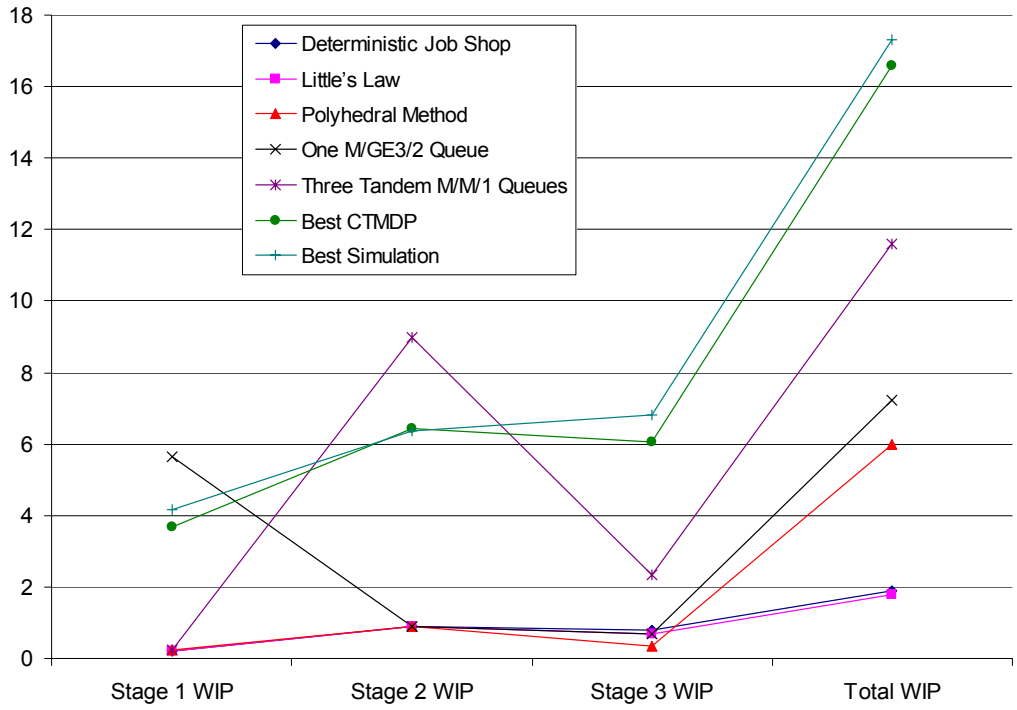

Figure 3.12: Comparison of Queue Levels Given by Different Models

4. Continuous Linear Programming (CLP) Models: Fluid Relaxation

In this chapter, we show for the first time that a relaxation of the deterministic discrete job shop (described in Chapter 2) produces a continuous linear programming (CLP) formulation similar to the fluid models that others have studied. Specifically, we relax the integrality requirement on the cumulative number of jobs that have finished processing $u_{j,k,m}(t)$, on the number of jobs in queue or in service $q_{j,k}(t)$, and on the arrival stream $\lfloor \alpha_{j,k}(t-t_0) \rfloor$. Thus, constraint (2.7) can be deleted, and constraint (2.6) becomes:

$$\tilde{q}_{j,k}(t) \geq 0 \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \end{cases}, \tag{4.1}$$

where the overscript tilde "~" above a decision variable indicates that it applies to the fluid relaxation problem only. Constraint (2.4) becomes:

$$\begin{aligned} &\sum_{m=1}^{M_{t(j,1)}} \tilde{u}_{j,1,m}(t) + \tilde{q}_{j,1}(t) \\ &= \sum_{m=1}^{M_{t(j,k)}} u_{j,1,m}(t_0) + q_{j,1}(t_0) + (t-t_0)\alpha_{j,1} \end{aligned} \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \end{cases}, \tag{4.2}$$

and constraint (2.5) becomes:

$$\begin{aligned} &\sum_{m=1}^{M_{t(j,k)}} \tilde{u}_{j,k,m}(t) - \sum_{m=1}^{M_{t(j,k-1)}} \tilde{u}_{j,k-1,m}(t) + \tilde{q}_{j,k}(t) \\ &= \sum_{m=1}^{M_{t(j,k)}} u_{j,k,m}(t_0) - \sum_{m=1}^{M_{t(j,k-1)}} u_{j,k-1,m}(t_0) + q_{j,k}(t_0) \\ &+ (t-t_0)\alpha_{j,k} \end{aligned} \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}. \tag{4.3}$$

Now that both terms in constraint (2.8) can take on non-integer values, we may make it a binding equality,

$$\frac{\tau_{j,k,m}(t)}{p_{j,k}} - \tilde{u}_{j,k,m}(t) = 0 \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \\ \forall m \in \{1,2,...,M_{\iota(j,k)}\} \end{cases} . \tag{4.4}$$

We can do this because the only equations affecting $\tau_{j,k,m}(t)$ bound it from above while no mechanism drives it upward. Meanwhile, the objective functions drive $\tilde{u}_{j,k,m}(t)$ to increase, so we may assume equality without any loss of generality. To simplify, we can delete equation (4.4) and replace all occurrences of $\tau_{j,k,m}(t)$ with $p_{j,k}\tilde{u}_{j,k,m}(t)$ throughout the formulation. Thus, constraint (2.9) becomes:

$$\sum_{(j,k)\in\sigma_i} \langle 0 > 0 \rangle \leq 1 \qquad \begin{cases} \forall t \geq t_0 \\ \forall i \in \{1,2,...,I\} \\ \forall m \in \{1,2,...,M_i\} \end{cases} ,$$

which is always feasible, so it can be deleted. Constraint (2.10) becomes:

$$\sum_{m=1}^{M_{\iota(j,k)}} \tilde{u}_{j,k,m}(t) - \sum_{m=1}^{M_{\iota(j,k-1)}} \tilde{u}_{j,k-1,m}(t) \leq q_{j,k}(t_0) \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases} , \tag{4.5}$$

constraint (2.11) becomes:

$$\tilde{u}_{j,k,m}(t') - \tilde{u}_{j,k,m}(t) \geq 0 \qquad \begin{cases} \forall t,t' : t_0 \leq t < t' \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \\ \forall m \in \{1,2,...,M_{\iota(j,k)}\} \end{cases} , \tag{4.6}$$

and constraint (2.12) becomes:

$$\sum_{(j,k)\in\sigma_i} p_{j,k}\left[\tilde{u}_{j,k,m}(t')-\tilde{u}_{j,k,m}(t)\right]\le t'-t \qquad \begin{cases}\forall t,t': t_0\le t<t'\\ \forall i\in\{1,2,...,I\}\\ \forall m\in\{1,2,...,M_i\}\end{cases}. \tag{4.7}$$

As a further relaxation, we no longer distinguish between different machines of the same type. Thus, we sum constraints (4.6), and (4.7) over the M_i machines of each type to get:

$$\tilde{u}_{j,k}(t')-\tilde{u}_{j,k}(t)\ge 0 \qquad \begin{cases}\forall t,t': t_0\le t<t'\\ \forall j\in\{1,2,...,J\}\\ \forall k\in\{1,2,...,K_j\}\end{cases}, \tag{4.8}$$

and

$$\sum_{(j,k)\in\sigma_i} p_{j,k}\left[\tilde{u}_{j,k}(t')-\tilde{u}_{j,k}(t)\right]\le (t'-t)M_i \qquad \begin{cases}\forall t,t': t_0\le t<t'\\ \forall i\in\{1,2,...,I\}\end{cases}. \tag{4.9}$$

However, it would be more useful to replace M_i (the quantity of each machine type) with $a_i M_i$ (the average number of available machines of each machine type) which includes the average availability a_t as a factor. Thus, constraint (4.9) becomes:

$$\sum_{(j,k)\in\sigma_i} p_{j,k}\left[\tilde{u}_{j,k}(t')-\tilde{u}_{j,k}(t)\right]\le (t'-t)a_i M_i \qquad \begin{cases}\forall t,t': t_0\le t<t'\\ \forall i\in\{1,2,...,I\}\end{cases}. \tag{4.10}$$

In constraints (4.2), (4.3), and (4.5), we replace $\sum_{m=1}^{M_{\iota(j,k)}} u_{j,k,m}(t)$ with $u_{j,k}(t)$ to get (respectively)

$$\tilde{u}_{j,1}(t)+\tilde{q}_{j,1}(t)=u_{j,1}(t_0)+q_{j,1}(t_0)+\left(t-t_0\right)\alpha_{j,1} \qquad \begin{cases}\forall t\ge t_0\\ \forall j\in\{1,2,...,J\}\end{cases}, \tag{4.11}$$

$$\begin{aligned}&\tilde{u}_{j,k}(t)-\tilde{u}_{j,k-1}(t)+\tilde{q}_{j,k}(t)\\&=u_{j,k}(t_0)-u_{j,k-1}(t_0)+q_{j,k}(t_0)+(t-t_0)\alpha_{j,k}\end{aligned}\qquad\begin{cases}\forall t\geq t_0\\\forall j\in\{1,2,...,J\}\\\forall k\in\{2,3,...,K_j\}\end{cases},\tag{4.12}$$

and

$$\tilde{u}_{j,k}(t)-\tilde{u}_{j,k-1}(t)\leq q_{j,k}(t_0)\qquad\begin{cases}\forall t\geq t_0\\\forall j\in\{1,2,...,J\}\\\forall k\in\{2,3,...,K_j\}\end{cases}.\tag{4.13}$$

Constraint (4.13) is now redundant with respect to constraints (4.1) and (4.12), and may be deleted.

In order to show that the maximum workload objective function is equivalent to a linear function, we introduce two new variables: the immediate workload (the time to process all jobs currently waiting) at machine *i,*

$$w_i(t)\equiv\frac{\sum_{(j,k)\in\sigma_i}p_{j,k}q_{j,k}(t)}{a_iM_i}\qquad\forall i\in\{1,2,...,I\},$$

and the maximum immediate workload over all machines,

$$w_{\max}(t)\equiv\max\{w_i(t):i=1,2,\ldots,I\}.$$

The maximum workload objective function (2.3) then becomes

$$\tilde{C}_{\mathrm{w}}(\tilde{\mathbf{q}},\tilde{\mathbf{u}},\tilde{w}_{\max}\mid T,...)=\int_{t_0}^{t_0+T}\tilde{w}_{\max}(t)dt,\tag{4.14}$$

and we introduce a new constraint

$$\tilde{w}_{\max}(t)-\frac{\sum_{(j,k)\in\sigma_i}p_{j,k}\tilde{q}_{j,k}(t)}{a_iM_i}\geq 0\qquad\begin{cases}\forall t\geq t_0\\\forall i\in\{1,2,...,I\}\end{cases}.\tag{4.15}$$

Putting all the remaining equations together gives Formulation 4.1, a continuous linear program (CLP), although the makespan objective function (2.1) is not linear.

Formulation 4.1: Continuous Linear Program (CLP) for Fluid Relaxation

$$\min_{\tilde{\mathbf{q}},\tilde{\mathbf{u}}} \tilde{C}_{\max}(\tilde{\mathbf{q}},\tilde{\mathbf{u}} \mid T,...) = \int_{t_0}^{t_0+T} \left\langle \sum_{j=1}^{J}\sum_{k=1}^{K_j} \tilde{q}_{j,k}(t) > 0 \right\rangle dt \quad \text{or}$$

$$\min_{\tilde{\mathbf{q}},\tilde{\mathbf{u}}} \tilde{C}_{\mathrm{h}}(\tilde{\mathbf{q}},\tilde{\mathbf{u}} \mid T,...) = \int_{t_0}^{t_0+T} \sum_{j=1}^{J}\sum_{k=1}^{K_j} c_{j,k}\tilde{q}_{j,k}(t)dt \quad \text{or}$$

$$\min_{\tilde{\mathbf{q}},\tilde{\mathbf{u}},\tilde{w}_{\max}} \tilde{C}_{\mathrm{w}}(\tilde{\mathbf{q}},\tilde{\mathbf{u}},\tilde{w}_{\max} \mid T,...) = \int_{t_0}^{t_0+T} \tilde{w}_{\max}(t)dt$$

s.t.

$$\tilde{u}_{j,1}(t) + \tilde{q}_{j,1}(t) = u_{j,1}(t_0) + q_{j,1}(t_0) + (t-t_0)\alpha_{j,1} \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \end{cases}$$

$$\begin{aligned} &\tilde{u}_{j,k}(t) - \tilde{u}_{j,k-1}(t) + \tilde{q}_{j,k}(t) \\ &= u_{j,k}(t_0) - u_{j,k-1}(t_0) + q_{j,k}(t_0) + (t-t_0)\alpha_{j,k} \end{aligned} \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}$$

$$\tilde{u}_{j,k}(t') - \tilde{u}_{j,k}(t) \geq 0 \qquad \begin{cases} \forall t,t' : t_0 \leq t < t' \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \end{cases}$$

$$\sum_{(j,k)\in\sigma_i} p_{j,k}\left[\tilde{u}_{j,k}(t') - \tilde{u}_{j,k}(t)\right] \leq (t'-t)a_i M_i \qquad \begin{cases} \forall t,t' : t_0 \leq t < t' \\ \forall i \in \{1,2,...,I\} \end{cases}$$

$$\tilde{w}_{\max}(t) - \frac{\sum_{(j,k)\in\sigma_i} p_{j,k}\tilde{q}_{j,k}(t)}{a_i M_i} \geq 0 \qquad \begin{cases} \forall t \geq t_0 \\ \forall i \in \{1,2,...,I\} \end{cases}$$

$$\tilde{q}_{j,k}(t) \geq 0 \qquad \begin{cases} \forall t \geq t_0 \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \end{cases}$$

The entire CLP formulation can be put into matrix-vector form. The makespan objective function (2.1) becomes

$$\tilde{C}_{\max}(\tilde{\mathbf{q}},\tilde{\mathbf{u}}\,|\,T,...) = \int_{t_0}^{t_0+T} \left\langle \left\|\tilde{\mathbf{q}}(t)\right\|_1 > 0 \right\rangle dt\,,$$

where any norm $\|\cdot\|$ could have been used. The weighted holding cost objective function (2.2) becomes

$$\tilde{C}_{\mathrm{h}}(\tilde{\mathbf{q}},\tilde{\mathbf{u}}\,|\,T,...) = \int_{t_0}^{t_0+T} \mathbf{c}^{\mathrm{T}}\tilde{\mathbf{q}}(t)dt\,, \tag{4.16}$$

and the maximum workload objective function (4.14) remains unchanged.

Constraints (4.8) and (4.10) become

$$\tilde{\mathbf{u}}(t') - \tilde{\mathbf{u}}(t) \geq \mathbf{0}_{K,1} \qquad \forall t,t' : t_0 \leq t < t'\,, \tag{4.17}$$

and

$$\mathbf{BD}(\mathbf{p})\left[\tilde{\mathbf{u}}(t') - \tilde{\mathbf{u}}(t)\right] \leq (t'-t)\mathbf{D}(\mathbf{a})\mathbf{m} \qquad \forall t,t' : t_0 \leq t < t'\,. \tag{4.18}$$

Similarly, constraint (4.15) becomes

$$\mathbf{1}_I\,\tilde{w}_{\max}(t) - \left[\mathbf{D}(\mathbf{a})\mathbf{D}(\mathbf{m})\right]^{-1}\mathbf{BD}(\mathbf{p})\mathbf{q}(t) \geq 0 \qquad \forall t \geq t_0\,. \tag{4.19}$$

Also, constraint (4.1) becomes

$$\tilde{\mathbf{q}}(t) \geq \mathbf{0}_{K,1} \qquad \forall t \geq t_0\,. \tag{4.20}$$

Finally, if we merge constraints (4.11) and (4.12), we get

$$\begin{aligned}&\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\tilde{\mathbf{u}}(t) + \tilde{\mathbf{q}}(t) \\ &= \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\mathbf{u}(t_0) + \mathbf{q}(t_0) + \left(t - t_0\right)\boldsymbol{\alpha}\end{aligned} \qquad \forall t \geq t_0\,. \tag{4.21}$$

Putting these equations together gives Formulation 4.2 in matrix-vector form which is equivalent to Formulation 4.1.

Formulation 4.2: Continuous Linear Program (CLP) for Fluid Relaxation (in Matrix-Vector Form)

$$\min_{\tilde{\mathbf{q}},\tilde{\mathbf{u}}} \tilde{C}_{\max}(\tilde{\mathbf{q}},\tilde{\mathbf{u}} \mid T,...) = \int_{t_0}^{t_0+T} \left\langle \left\| \tilde{\mathbf{q}}(t) \right\|_1 > 0 \right\rangle dt \quad \text{or}$$

$$\min_{\tilde{\mathbf{q}},\tilde{\mathbf{u}}} \tilde{C}_{\mathrm{h}}(\tilde{\mathbf{q}},\tilde{\mathbf{u}} \mid T,...) = \int_{t_0}^{t_0+T} \mathbf{c}^{\mathrm{T}} \tilde{\mathbf{q}}(t) dt \quad \text{or}$$

$$\min_{\tilde{\mathbf{q}},\tilde{\mathbf{u}},\tilde{w}_{\max}} \tilde{C}_{\mathrm{w}}(\tilde{\mathbf{q}},\tilde{\mathbf{u}},\tilde{w}_{\max} \mid T,...) = \int_{t_0}^{t_0+T} \tilde{w}_{\max}(t) dt$$

$$\text{s.t.} \quad \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\tilde{\mathbf{u}}(t) + \tilde{\mathbf{q}}(t) = \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\mathbf{u}(t_0) + \mathbf{q}(t_0) + \left(t - t_0\right)\boldsymbol{\alpha} \qquad \forall t \geq t_0$$

$$\tilde{\mathbf{u}}(t') - \tilde{\mathbf{u}}(t) \geq \mathbf{0}_{K,1} \qquad \forall t, t' : t_0 \leq t < t'$$

$$\mathbf{BD}(\mathbf{p})\left[\tilde{\mathbf{u}}(t') - \tilde{\mathbf{u}}(t)\right] \leq (t' - t)\mathbf{D}(\mathbf{a})\mathbf{m} \qquad \forall t, t' : t_0 \leq t < t'$$

$$\mathbf{1}_I \tilde{w}_{\max}(t) - \left[\mathbf{D}(\mathbf{a})\mathbf{D}(\mathbf{m})\right]^{-1} \mathbf{BD}(\mathbf{p})\mathbf{q}(t) \geq 0 \qquad \forall t \geq t_0$$

$$\tilde{\mathbf{q}}(t) \geq \mathbf{0}_{K,1} \qquad \forall t \geq t_0$$

It should be noted here that although we use the word "fluid" to describe this model, we do not mean to imply that we are using the laws of fluid dynamics such as solving the Navier-Stokes equations. Instead, we merely wish to convey the idea of a continuously flowing product moving in a liquid-like fashion through the factory (with no concern for such things as turbulence). Each queue may be pictured as a holding tank, while each machine may be visualized as a set of pumps (one for each queue from which it draws) that share a finite power source. The processing time at each stage may be thought of as being in proportion to the height of the pipe leading to the next holding tank. This analogy quickly breaks

down, since we have the fluid moving through the pipes instantaneously, but we hope the mental picture (like any model) gives insight.

4.1 INSENSITIVITY OF CLP FORMULATION TO TOTAL INITIAL WIP

It has been shown by others that the CLP formulation is not sensitive to the total initial WIP $\|\mathbf{q}(t_0)\|_1$ but only to the relative amounts of initial WIP in each buffer $\mathbf{q}(t_0)/\|\mathbf{q}(t_0)\|_1$. To see this, consider a positive scale factor ν (the Greek letter nu). Without loss of generality, assume $t_0 = 0$. For a given set of problem data, scale the initial values and T by ν, so $\mathbf{q}(t_0) \to \nu\mathbf{q}(0)$, $\mathbf{u}(t_0) \to \nu\mathbf{u}(0)$, and $T \to \nu T$. In the remainder of the formulation, we can (without loss of generality) replace the decision variables with their scaled versions, so $\tilde{\mathbf{q}}(t) \to \nu\tilde{\mathbf{q}}(t/\nu)$, $\tilde{\mathbf{u}}(t) \to \nu\tilde{\mathbf{u}}(t/\nu)$, and $\tilde{w}_{\max}(t) \to \nu\tilde{w}_{\max}(t/\nu)$.

The constraints (4.17) through (4.21) become

$$\begin{aligned}&\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\nu\tilde{\mathbf{u}}\left(\frac{t}{\nu}\right) + \nu\tilde{\mathbf{q}}\left(\frac{t}{\nu}\right) && \forall t : t \ge 0,\\ &= \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\nu\mathbf{u}(0) + \nu\mathbf{q}(0) + (t-0)\boldsymbol{\alpha}\end{aligned}$$

$$\mathbf{BD}(\mathbf{p})\left[\nu\tilde{\mathbf{u}}\left(\frac{t'}{\nu}\right) - \nu\tilde{\mathbf{u}}\left(\frac{t}{\nu}\right)\right] \le (t'-t)\mathbf{D}(\mathbf{a})\mathbf{m} \qquad \forall t, t' : 0 \le t < t',$$

$$\nu\tilde{\mathbf{u}}\left(\frac{t'}{\nu}\right) - \nu\tilde{\mathbf{u}}\left(\frac{t}{\nu}\right) \ge \mathbf{0}_{K,1} \qquad \forall t, t' : 0 \le t < t',$$

$$\mathbf{1}_I \nu\tilde{w}_{\max}\left(\frac{t}{\nu}\right) - \left[\mathbf{D}(\mathbf{a})\mathbf{D}(\mathbf{m})\right]^{-1}\mathbf{BD}(\mathbf{p})\nu\tilde{\mathbf{q}}\left(\frac{t}{\nu}\right) \ge 0 \quad \forall t : t \ge 0,$$

and

$$\nu\tilde{\mathbf{q}}\left(\frac{t}{\nu}\right) \geq \mathbf{0}_{K,1} \qquad \forall t : t \geq 0 .$$

If we divide all the constraints by ν and (without loss of generality) rescale the time limits (so $t \to t/\nu$), we get the following equations which are equivalent to the original constraints (4.17) through (4.21) when $t_0 = 0$.

$$\begin{aligned} &\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\tilde{\mathbf{u}}\left(\frac{t}{\nu}\right) + \tilde{\mathbf{q}}\left(\frac{t}{\nu}\right) \\ &= \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\mathbf{u}(0) + \mathbf{q}(0) + \left(\frac{t}{\nu} - 0\right)\boldsymbol{\alpha} \end{aligned} \qquad \forall t : \frac{t}{\nu} \geq 0 ,$$

$$\mathbf{BD}(\mathbf{p})\left[\tilde{\mathbf{u}}\left(\frac{t'}{\nu}\right) - \tilde{\mathbf{u}}\left(\frac{t}{\nu}\right)\right] \leq \left(\frac{t'}{\nu} - \frac{t}{\nu}\right)\mathbf{D}(\mathbf{a})\mathbf{m} \qquad \forall t, t' : 0 \leq \frac{t}{\nu} < \frac{t'}{\nu} ,$$

$$\tilde{\mathbf{u}}\left(\frac{t'}{\nu}\right) - \tilde{\mathbf{u}}\left(\frac{t}{\nu}\right) \geq \mathbf{0}_{K,1} \qquad \forall t, t' : 0 \leq \frac{t}{\nu} < \frac{t'}{\nu} ,$$

$$\mathbf{1}_I \tilde{w}_{\max}\left(\frac{t}{\nu}\right) - \left[\mathbf{D}(\mathbf{a})\mathbf{D}(\mathbf{m})\right]^{-1}\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}\left(\frac{t}{\nu}\right) \geq 0 \qquad \forall t : \frac{t}{\nu} \geq 0 ,$$

and

$$\tilde{\mathbf{q}}\left(\frac{t}{\nu}\right) \geq \mathbf{0}_{K,1} \qquad \forall t : \frac{t}{\nu} \geq 0 .$$

An appropriate transformation of variables ($t' = t/\nu$) shows how the new objective functions are related to the original ones:

$$\begin{aligned} \tilde{C}_{\max}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \middle| \nu T, \nu\mathbf{q}(0), \nu\mathbf{u}(0), \ldots\right) &= \int_0^{0+\nu T} \left\langle \left\| \nu\tilde{\mathbf{q}}\left(\frac{t}{\nu}\right) \right\|_1 > 0 \right\rangle dt \\ &= \int_0^T \left\langle \left\| \tilde{\mathbf{q}}(t') \right\|_1 > 0 \right\rangle \nu\, dt' \end{aligned}$$

$$= \nu \tilde{C}_{\max}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right).$$

$$\tilde{C}_{\mathrm{h}}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \middle| \nu T, \nu\mathbf{q}(0), \nu\mathbf{u}(0), \ldots\right) = \int_0^{0+\nu T} \mathbf{c}^{\mathrm{T}}\left[\nu\tilde{\mathbf{q}}\left(\frac{t}{\nu}\right)\right] dt$$

$$= \int_0^{T} \nu \mathbf{c}^{\mathrm{T}} \tilde{\mathbf{q}}(t') \nu dt'$$

$$= \nu^2 \tilde{C}_{\mathrm{h}}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right).$$

$$C_{\mathrm{w}}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}}, \tilde{w}_{\max} \middle| \nu T, \nu\mathbf{q}(0), \nu\mathbf{u}(0), \ldots\right) = \int_0^{0+\nu T} \nu \tilde{w}_{\max}\left(\frac{t}{\nu}\right) dt$$

$$= \int_0^{T} \nu \tilde{w}_{\max}(t') \nu dt'$$

$$= \nu^2 \tilde{C}_{\mathrm{w}}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}}, \tilde{w}_{\max} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right).$$

Thus, consider an original problem with data $\mathbf{q}(0)$, $\mathbf{u}(0)$, and T that has an optimal solution $\left\{\tilde{\mathbf{q}}^*(t), \tilde{\mathbf{u}}^*(t) : \forall t > 0\right\}$ with respective optimal objective function values given by $\tilde{C}^*_{\max}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right)$, $\tilde{C}^*_{\mathrm{h}}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right)$, or $\tilde{C}^*_{\mathrm{w}}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}}, \tilde{w}_{\max} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right)$. Then a new problem with data $\nu\mathbf{q}(0)$, $\nu\mathbf{u}(0)$, and νT would have an optimal solution $\left\{\nu\tilde{\mathbf{q}}^*(t/\nu), \nu\tilde{\mathbf{u}}^*(t/\nu) : \forall t > 0\right\}$ with respective optimal objective function values given by $\nu\tilde{C}^*_{\max}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right)$, $\nu^2\tilde{C}^*_{\mathrm{h}}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right)$, or $\nu^2\tilde{C}^*_{\mathrm{w}}\left(\tilde{\mathbf{q}}, \tilde{\mathbf{u}}, \tilde{w}_{\max} \middle| T, \mathbf{q}(0), \mathbf{u}(0), \ldots\right)$.

If we denote the equivalent (possibly non-integer) number of machines working on jobs of type *j* in stage *k* at time *t* by

$$v_{j,k}(t) \equiv p_{j,k}\dot{u}_{j,k}(t),$$

(the Latin letter "vee" not the Greek letter "nu") or, in vector form

$\mathbf{v}(t) \equiv \mathbf{D}(\mathbf{p})\dot{\mathbf{u}}(t)$,

then $\tilde{\mathbf{v}}^*(0^+)$ (the optimal initial allocation of machines to buffers) is the same for both the original problem and new problem. Thus, $\tilde{\mathbf{v}}^*(0^+)$ is completely determined by $\mathbf{q}(t_0)/\|\mathbf{q}(t_0)\|_1$ with no additional dependence on $\|\mathbf{q}(t_0)\|_1$.

The practical implication of all this is that an online policy $\hat{\mathbf{v}}(t_0^+)$ based on the optimal fluid solution need not be computed for all $\mathbf{q}(t_0) \in \mathbb{Z}_+^K$. It may suffice to compute $\hat{\mathbf{v}}(t_0^+)$ for a set of $\mathbf{q}(t_0)$ vectors that span the $(K-1)$-dimensional simplex $\left\{\mathbf{q}(t_0) \in \mathbb{R}_+^K : \|\mathbf{q}(t_0)\|_1 = 1\right\}$ and then interpolate for any other $\mathbf{q}(t_0)/\|\mathbf{q}(t_0)\|_1$ vector needed. However, $\tilde{\mathbf{v}}^*(0^+)$ is not necessarily a linear (or even continuous) function of $\mathbf{q}(t_0)/\|\mathbf{q}(t_0)\|_1$, so such interpolations give only an approximation of an optimal solution to the fluid problem. On the other hand, the fluid problem is a relaxation of the discrete problem, so an optimal solution to the fluid problem may not be any more useful.

4.2 Optimal Solution to the Fluid Makespan Objective

Bertsimas and Sethuraman [2002] showed that if the system obeys the stability condition

$$\rho_i \equiv \frac{\sum_{(j,k)\in\sigma_i} p_{j,k}\alpha_{j,k}^+}{a_i M_i} < 1 \qquad \forall i \in \{1,2,...,I\},$$

then we have a finite $\theta_i(t)$, the machine congestion (the mean fraction of time required for machine *i* to process all of its tasks that are either present at time t_0 or that arrive by time *t*):

$$\theta_i(t) \equiv \frac{\sum_{(j,k)\in\sigma_i} p_{j,k}\left[q^+_{j,k}(t_0) + \alpha^+_{j,k}\left(t - t_0\right)\right]}{a_i M_i\left(t - t_0\right)} < \infty \qquad \begin{cases} \forall t \geq t_0 \\ \forall i \in \{1,2,...,I\} \end{cases}.$$

The bottleneck congestion is then given by

$$\theta_{\max}(t) \equiv \max\left\{\theta_i(t) : i = 1,2,\ldots,I\right\}.$$

Since

$$\sum_{j,k\in\sigma_i} \tilde{q}_{j,k}(t) > 0 \qquad \begin{cases} \forall t : t_0 \leq t \leq t_0 + \theta_i\left(t\right) \\ \forall i \in \{1,2,...,I\} \end{cases},$$

we know that

$$\tilde{C}^*_{\max}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \mid T) \geq T \min\left\{1, \theta_{\max}(t_0 + T)\right\}.$$

By definition of the fluid makespan objective, we also know that

$$\tilde{C}^*_{\max}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \mid T) \leq T\ .$$

Let

$$T^* \equiv \max\left\{\frac{\sum_{(j,k)\in\sigma_i} p_{j,k} q^+_{j,k}(t_0)}{a_i M_i - \sum_{(j,k)\in\sigma_i} p_{j,k}\alpha^+_{j,k}} : i = 1,2,\ldots,I\right\}$$

$$= \left\|\left[\mathbf{D}(\mathbf{1}_I - \boldsymbol{\rho})\right]^{-1} \mathbf{w}(t_0)\right\|_\infty,$$

and let

$$i^* \equiv \arg\max\left\{\frac{\sum_{(j,k)\in\sigma_i} p_{j,k} q^+_{j,k}(t_0)}{a_i M_i - \sum_{(j,k)\in\sigma_i} p_{j,k}\alpha^+_{j,k}} : i = 1,2,\ldots,I\right\}.$$

Then

$$\theta_{i^*}(t_0 + T^*) = \frac{\sum_{(j,k)\in\sigma_{i^*}} p_{j,k}\left[q^+_{j,k}(t_0) + \alpha^+_{j,k}T^*\right]}{a_{i^*}M_{i^*}T^*}$$

$$= \frac{\sum_{(j,k)\in\sigma_{i^*}} p_{j,k}\alpha^+_{j,k}}{a_{i^*}M_{i^*}} + \frac{\sum_{(j,k)\in\sigma_{i^*}} p_{j,k}q^+_{j,k}(t_0)}{a_{i^*}M_{i^*}T^*}$$

$$= \rho_{i^*} + \frac{\sum_{(j,k)\in\sigma_{i^*}} p_{j,k}q^+_{j,k}(t_0)}{a_{i^*}M_{i^*}\dfrac{a_{i^*}M_{i^*} - \sum_{(j,k)\in\sigma_{i^*}} p_{j,k}\alpha^+_{j,k}}{a_{i^*}M_{i^*}}}$$

$$= \rho_{i^*} + \frac{a_{i^*}M_{i^*} - \sum_{(j,k)\in\sigma_{i^*}} p_{j,k}\alpha^+_{j,k}}{a_{i^*}M_{i^*}}$$

$$= \rho_{i^*} + 1 - \rho_{i^*}$$

$$= 1.$$

Thus

$$T^* = T^* \min\left\{1, \theta_{\max}(t_0 + T^*)\right\} \le \tilde{C}^*_{\max}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \mid T^*, ...) \le T^*,$$

so

$$\tilde{C}^*_{\max}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \mid T^*, ...) = T^*.$$

Bertsimas and Sethuraman [2002] showed that the following solution attains this optimal fluid makespan objective (not necessarily uniquely) $\forall T \ge T^*$:

$$\tilde{q}^{*}_{j,k}(t) = \begin{cases} q_{j,k}(t_0)\left(1 - \dfrac{t - t_0}{T^*}\right) & \text{if } t_0 \le t \le t_0 + T^* \\ 0 & \text{if } t_0 + T^* \le t \end{cases},$$

$$\tilde{q}^{+*}_{j,k}(t) = \begin{cases} q^{+}_{j,k}(t_0)\left(1 - \dfrac{t - t_0}{T^*}\right) & \text{if } t_0 \le t \le t_0 + T^* \\ 0 & \text{if } t_0 + T^* \le t \end{cases},$$

$$\tilde{u}^{*}_{j,k}(t) = \begin{cases} u_{j,k}(t_0) + \left(t - t_0\right)\left[\dfrac{q^{+}_{j,k}(t_0)}{T^*} + \alpha^{+}_{j,k}\right] & \text{if } t_0 \le t \le t_0 + T^* \\ u_{j,k}(t_0) + q^{+}_{j,k}(t_0) + \alpha^{+}_{j,k}\left(t - t_0\right) & \text{if } t_0 + T^* \le t \le t_0 + T \\ u_{j,k}(t_0) + q^{+}_{j,k}(t_0) + \alpha^{+}_{j,k}T & \text{if } t_0 + T \le t \end{cases}.$$

Thus, the equivalent (possibly non-integer) number of machines working on jobs of type j in stage k at time t is

$$\begin{aligned} \tilde{v}^{*}_{j,k}(t) &\equiv p_{j,k}\dot{\tilde{u}}^{*}_{j,k}(t) \\ &= \begin{cases} p_{j,k}\left[\dfrac{q_{j,k}(t_0)}{T^*} + \alpha^{+}_{j,k}\right] & \text{if } t_0 \le t \le t_0 + T^* \\ p_{j,k}\alpha^{+}_{j,k} & \text{if } t_0 + T^* \le t \le t_0 + T \\ 0 & \text{if } t_0 + T \le t \end{cases}. \end{aligned}$$

Since

$$\tilde{q}^{*}_{j,k}(t) = 0 \qquad \forall t \ge t_0 + T^*,$$

we know that

$$\tilde{C}^{*}_{\max}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \mid T, \ldots) = T^* \qquad \forall T \ge T^*.$$

Although this solution (that attains the optimal fluid makespan objective) is well known, we discuss it here and in the following section because it is used as a starting point when other objective functions are used.

4.3 EXAMPLE NETWORK: OPTIMAL MAKESPAN

The optimal fluid makespan solution is plotted in the following figures for the example network. Figure 4.1 shows the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ over time. Note that all of the values of $\tilde{q}^{+*}_{j,k}(t)$ drop linearly from their initial value $q^{+}_{j,k}(t_0)$ (at time $t_0 = 0$) to zero at time $\tilde{C}^{*}_{\max}(T) = T^{*} =$ 128 minutes.

The thick black line (which has slope $\alpha_{1,1}$) indicates the path followed by an infinitesimal bit of fluid that enters the network at time zero. The height of the thick black line at each point in time represents the amount of fluid that entered the system after that infinitesimal bit of fluid, and the time that the thick black line crosses the other lines is the time at which that infinitesimal bit of fluid leaves one buffer and enters another (or leaves the system). Fluid already in the system at time zero begins at the appropriate height on the vertical axis and follows a path parallel to the thick black line. Fluid that enters the system at a later time begins at the appropriate distance along the horizontal axis and also follows a path parallel to the thick black line.

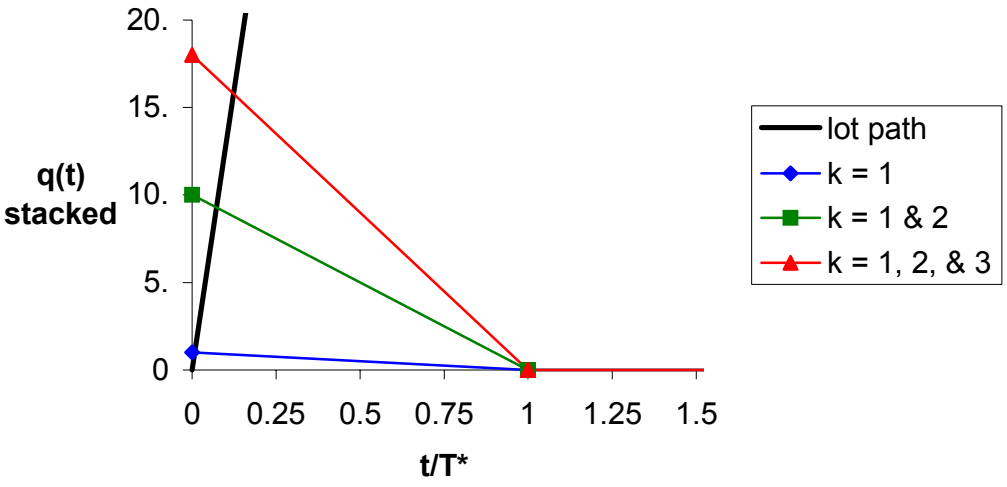

Figure 4.1: Total WIP for Optimal Makespan in Example Network

Figure 4.2 shows the different elements of $\tilde{v}^*_{j,k}(t)$ stacked upon each other (for each machine type) to show what fraction of its capacity the machine type devotes to each queue over time. Machine 1 is the bottleneck (which determines T^*), so its available capacity is fully utilized until time T^*. The other machine operates at less that its available capacity. After time T^*, both machines only operate at a level sufficient to process the ongoing inputs (which arrive at rate $\alpha_{1,1}$).

Figure 4.3 shows an important result of the fluid model: $\tilde{u}^*_{j,k}(t)$, the cumulative number of units that have left each queue over time. These are the values that the associated heuristic scheduling method will attempt to match.

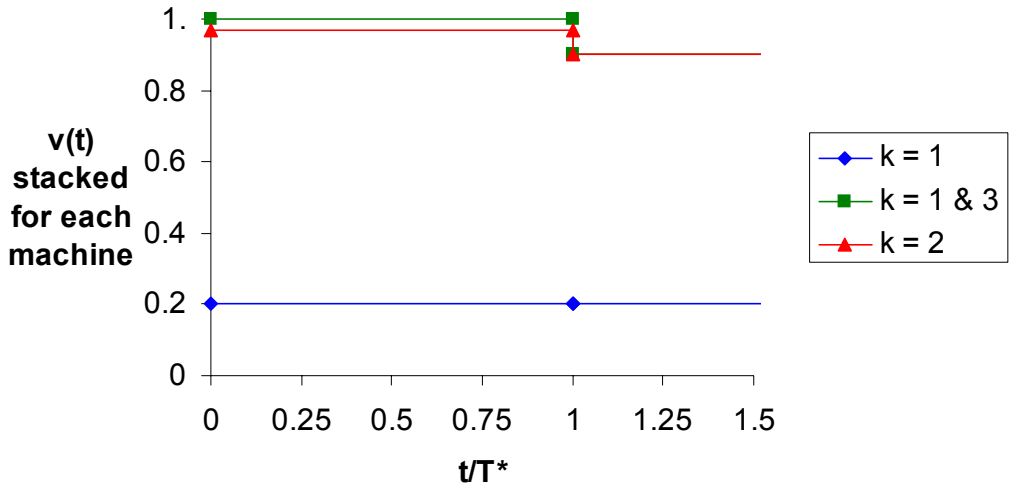

Figure 4.2: Machine Utilization for Optimal Makespan in Example Network

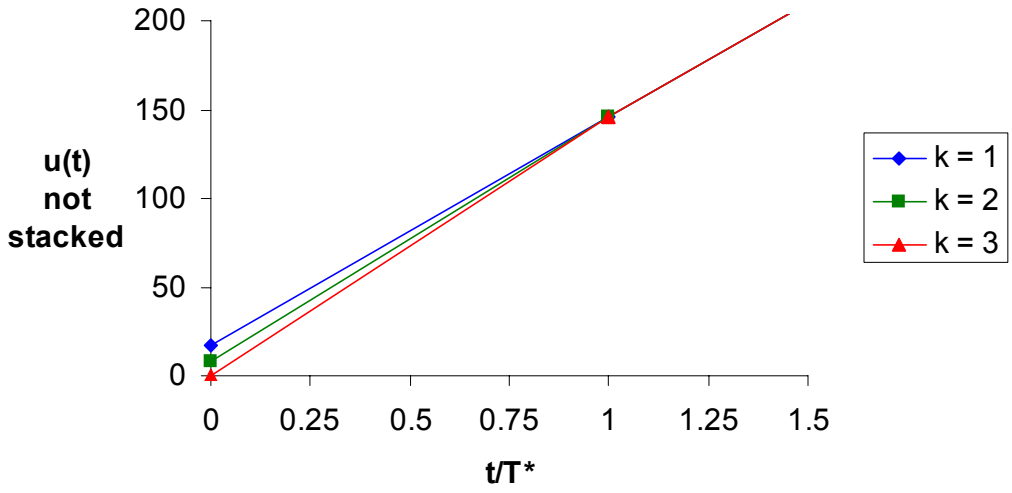

Figure 4.3: Cumulative Units Processed for Optimal Makespan in Example Network

5. Separated Continuous Linear Programming (SCLP) Models: Fluid Relaxation

Weiss [2002] showed that an equivalent formulation to the CLP (with the weighted holding cost objective function only) can be derived by using the derivative $\dot{\tilde{\mathbf{u}}}(t)$ as a decision variable instead of $\tilde{\mathbf{u}}(t)$ which is absolutely continuous in the fluid relaxation (so its derivatives with respect to time exist almost everywhere). If we divide both sides of constraints (4.17) and (4.18) by $(t'-t)$, we get

$$\frac{\tilde{\mathbf{u}}(t')-\tilde{\mathbf{u}}(t)}{t'-t} \geq \mathbf{0}_{K,1} \qquad \forall t,t' : t_0 \leq t < t',$$

and

$$\mathbf{BD}(\mathbf{p})\frac{\tilde{\mathbf{u}}(t')-\tilde{\mathbf{u}}(t)}{t'-t} \leq \mathbf{D}(\mathbf{a})\mathbf{m} \qquad \forall t,t' : t_0 \leq t < t'.$$

Now letting $t' \searrow t$, we have the following constraints that hold almost everywhere:

$$\dot{\tilde{\mathbf{u}}}(t) \geq \mathbf{0}_{K,1} \qquad \forall t \geq t_0, \tag{5.1}$$

and

$$\mathbf{BD}(\mathbf{p})\dot{\tilde{\mathbf{u}}}(t) + \dot{\tilde{\mathbf{y}}}(t) = \mathbf{D}(\mathbf{a})\mathbf{m} \qquad \forall t \geq t_0. \tag{5.2}$$

Here we have inserted the (possibly non-integer) number of idle machines $\dot{\tilde{\mathbf{y}}}(t)$ as a new slack variable into constraint (5.2), so

$$\dot{\tilde{\mathbf{y}}}(t) \geq \mathbf{0}_{I,1} \qquad \forall t \geq t_0. \tag{5.3}$$

Since

$$\tilde{\mathbf{u}}(t) = \mathbf{u}(t_0) + \int_{t_0}^{t} \dot{\tilde{\mathbf{u}}}(s)ds \qquad \forall t \geq t_0,$$

constraint (4.21) can be written as

$$\begin{aligned} &\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\left[\mathbf{u}(t_0) + \int_{t_0}^{t} \dot{\tilde{\mathbf{u}}}(s)ds\right] + \tilde{\mathbf{q}}(t) \\ &= \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\mathbf{u}(t_0) + \mathbf{q}(t_0) + \boldsymbol{\alpha}\left(t - t_0\right) \end{aligned} \qquad \forall t \geq t_0,$$

or

$$\tilde{\mathbf{q}}(t) = \mathbf{q}(t_0) + \boldsymbol{\alpha}\left(t - t_0\right) - \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\int_{t_0}^{t} \dot{\tilde{\mathbf{u}}}(s)ds \qquad \forall t \geq t_0, \tag{5.4}$$

Now if we substitute constraint (5.4) into the objective (4.16) and integrate by parts, we can see that

$$\begin{aligned} \tilde{C}_{\mathrm{h}}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \mid T, ...) &= \int_{t_0}^{t_0+T} \mathbf{c}^{\mathrm{T}} \tilde{\mathbf{q}}(t)dt \\ &= \int_{t_0}^{t_0+T} \mathbf{c}^{\mathrm{T}} \left[\mathbf{q}(t_0) + \boldsymbol{\alpha}\left(t - t_0\right) - \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\int_{t_0}^{t} \dot{\tilde{\mathbf{u}}}(s)ds\right] dt \\ &= \mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)\int_{t_0}^{t_0+T} dt + \mathbf{c}^{\mathrm{T}}\boldsymbol{\alpha}\int_{t_0}^{t_0+T} \left(t - t_0\right) dt - \mathbf{c}^{\mathrm{T}}\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\int_{t_0}^{t_0+T}\int_{t_0}^{t} \dot{\tilde{\mathbf{u}}}(s)ds\ dt \\ &= \mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)T + \mathbf{c}^{\mathrm{T}}\boldsymbol{\alpha}\frac{T^2}{2} \\ &\quad -\mathbf{c}^{\mathrm{T}}\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\left[\left(t - t_0\right)\int_{t_0}^{t} \dot{\tilde{\mathbf{u}}}(s)ds\Big|_{t=t_0}^{t_0+T} - \int_{t_0}^{t_0+T} \left(t - t_0\right)\dot{\tilde{\mathbf{u}}}(t)dt\right] \end{aligned}$$

$$= \mathbf{c}^{\mathrm{T}} \left[T\mathbf{q}(t_0) + \frac{T^2}{2}\boldsymbol{\alpha} \right] - \mathbf{c}^{\mathrm{T}} \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}} \right) \left[T \int_{t_0}^{t_0+T} \dot{\tilde{\mathbf{u}}}(s)ds - \int_{t_0}^{t_0+T} (t - t_0)\dot{\tilde{\mathbf{u}}}(t)dt \right]$$

$$= \mathbf{c}^{\mathrm{T}} \left[T\mathbf{q}(t_0) + \frac{T^2}{2}\boldsymbol{\alpha} \right] - \mathbf{c}^{\mathrm{T}} \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}} \right) \int_{t_0}^{t_0+T} (t_0 + T - t)\dot{\tilde{\mathbf{u}}}(t)dt .$$

Eliminating the terms that are constant (when T is fixed), we can create a new maximizing objective function that is equivalent to the weighted holding cost objective:

$$\max_{\dot{\tilde{\mathbf{u}}}} \tilde{C}'_{\mathrm{h}}(\dot{\tilde{\mathbf{u}}}, \tilde{\mathbf{q}}, \dot{\tilde{\mathbf{y}}} \mid T, ...) = \int_{t_0}^{t_0+T} (t_0 + T - t)\left(\mathbf{c}^{+}\right)^{\mathrm{T}} \dot{\tilde{\mathbf{u}}}(t)dt ,$$

where we have substituted in $\mathbf{c}^{+} \equiv \left(\mathbf{I}_K - \mathbf{P}\right)\mathbf{c}$ which we will call the keeping cost (since each element represents the marginal cost of keeping one unit of fluid in its current buffer instead of processing it and sending it to its next buffer).

Note that even if either the CLP optimal objective function value $\tilde{C}^{*}_{\mathrm{h}}(\mathbf{q}, \mathbf{u} \mid T, ...)$ or this new optimal objective function value $\tilde{C}'^{*}_{\mathrm{h}}(\dot{\tilde{\mathbf{u}}}, \tilde{\mathbf{q}}, \dot{\tilde{\mathbf{y}}} \mid T, ...)$ is bounded as T increases, the other may still be unbounded, because

$$\tilde{C}_{\mathrm{h}}(\mathbf{q}, \mathbf{u} \mid T, ...) + \tilde{C}'_{\mathrm{h}}(\dot{\tilde{\mathbf{u}}}, \tilde{\mathbf{q}}, \dot{\tilde{\mathbf{y}}} \mid T, ...) = \mathbf{c}^{\mathrm{T}} \left[T\mathbf{q}(t_0) + \frac{T^2}{2}\boldsymbol{\alpha} \right] \to \infty \text{ as } T \to \infty .$$

Along with the lower bound

$$\tilde{\mathbf{q}}(t) \geq \mathbf{0}_{K,1} \qquad \forall t \geq t_0 , \tag{5.5}$$

we have Formulation 5.1, a separated continuous linear program (SCLP) with linear data that is equivalent to the CLP formulation given previously.

Formulation 5.1: Separated Continuous Linear Program (SCLP) for Fluid Relaxation

$$\max_{\dot{\mathbf{u}}} \tilde{C}'_{\mathrm{h}}(\dot{\tilde{\mathbf{u}}}, \tilde{\mathbf{q}}, \dot{\tilde{\mathbf{y}}} \mid T, \ldots) = \int_{t_0}^{t_0+T} (t_0 + T - t)\left(\mathbf{c}^+\right)^{\mathrm{T}} \dot{\tilde{\mathbf{u}}}(t)dt$$

$$\text{s.t.} \quad \int_{t_0}^{t} \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\dot{\tilde{\mathbf{u}}}(s)ds + \tilde{\mathbf{q}}(t) = \mathbf{q}(t_0) + \boldsymbol{\alpha}\left(t - t_0\right) \qquad \forall t \geq t_0$$

$$\mathbf{BD}(\mathbf{p})\dot{\tilde{\mathbf{u}}}(t) + \dot{\tilde{\mathbf{y}}}(t) = \mathbf{D}(\mathbf{a})\mathbf{m} \qquad \forall t \geq t_0$$

$$\dot{\tilde{\mathbf{u}}}(t) \geq \mathbf{0}_{K,1} \qquad \forall t \geq t_0$$

$$\tilde{\mathbf{q}}(t) \geq \mathbf{0}_{K,1} \qquad \forall t \geq t_0$$

$$\dot{\tilde{\mathbf{y}}}(t) \geq \mathbf{0}_{I,1} \qquad \forall t \geq t_0$$

The term "separated" is used because the constraints are separated into two sets: the integrated constraint (5.4) and the instantaneous constraints (5.1), (5.2), (5.3), and (5.5). Here $\dot{\tilde{\mathbf{u}}}(t)$ may be thought of as the primary decision variable with $\tilde{\mathbf{q}}(t)$ and $\dot{\tilde{\mathbf{y}}}(t)$ as slack variables for their respective constraints (so they appear as objective function arguments but not under the "min").

Pullan [1996] showed that weak duality holds between SCLP formulation and Formulation 5.2, the dual continuous linear program (DCLP):

Formulation 5.2: Dual Continuous Linear Program (DCLP) for Fluid Relaxation

$$\min_{\boldsymbol{\pi},\boldsymbol{\xi}} \tilde{C}''_{\mathrm{h}}(\boldsymbol{\pi}, \boldsymbol{\xi}, \boldsymbol{\omega} \mid T, \ldots) = \mathbf{q}(t_0)^{\mathrm{T}}\boldsymbol{\pi}(t_0 + T) + \int_{t_0}^{t_0+T} \left[\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\pi}(t) + \mathbf{m}^{\mathrm{T}}\mathbf{D}(\mathbf{a})\boldsymbol{\xi}(t)\right]dt$$

$$\text{s.t.} \quad \left(\mathbf{I}_K - \mathbf{P}\right)\boldsymbol{\pi}(t) + \mathbf{D}(\mathbf{p})\mathbf{B}^{\mathrm{T}}\boldsymbol{\xi}(t) - \boldsymbol{\omega}(t) = \left(\mathbf{c}^+\right)\left(t - t_0\right) \qquad \forall t \geq t_0$$

$$\boldsymbol{\pi}(t_0) = \mathbf{0}_{K,1}$$

$$\boldsymbol{\pi}(t) \text{ nondecreasing} \qquad \forall t \geq t_0$$

$$\boldsymbol{\omega}(t) \geq \mathbf{0}_{K,1} \qquad \forall t \geq t_0$$

$$\boldsymbol{\xi}(t) \in \mathbb{R}^I \text{ (unrestricted)} \qquad \forall t \geq t_0$$

Complementary slackness has been defined [Pullan 1997] as:

$\tilde{q}_{j,k}(t) > 0 \Leftrightarrow \dot{\pi}_{j,k}(\cdot) = 0$ in a neighborhood of $t_0 + T - t$,

$\dot{\tilde{y}}_i(t) > 0 \Leftrightarrow \xi_i\left(t_0 + T - t\right) = 0$,

and

$\dot{\tilde{u}}_{j,k}(t) > 0 \Leftrightarrow \omega_{j,k}\left(t_0 + T - t\right) = 0$.

Weak duality implies that any pair of complementarily slack primal and dual feasible solutions are optimal.

By a process that might be vaguely thought of as "differentiating" SCLP and DCLP, Weiss [2001] obtained a primal linear program:

$$\max_{\dot{\tilde{\mathbf{u}}}} \quad \left(\mathbf{c}^+\right)^{\mathrm{T}} \dot{\tilde{\mathbf{u}}}$$

$$\text{s.t.} \quad \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\dot{\tilde{\mathbf{u}}} + \dot{\tilde{\mathbf{q}}} = \boldsymbol{\alpha}$$

$$\mathbf{BD(p)}\dot{\tilde{\mathbf{u}}} + \dot{\tilde{\mathbf{y}}} = \mathbf{D(a)m}$$

$$\dot{\tilde{\mathbf{u}}} \geq \mathbf{0}_{K,1}$$

$$\dot{\tilde{\mathbf{y}}} \geq \mathbf{0}_{I,1}$$

and its dual linear program:

$$\min_{\boldsymbol{\pi},\boldsymbol{\xi}} \quad \boldsymbol{\alpha}^{\mathrm{T}}\dot{\boldsymbol{\pi}} + \mathbf{m}^{\mathrm{T}}\mathbf{D(a)}\dot{\boldsymbol{\xi}}$$

$$\text{s.t.} \quad \left(\mathbf{I}_K - \mathbf{P}\right)\dot{\boldsymbol{\pi}} + \mathbf{D(p)}\mathbf{B}^{\mathrm{T}}\dot{\boldsymbol{\xi}} - \dot{\boldsymbol{\omega}} = \mathbf{c}^+$$

$$\dot{\boldsymbol{\pi}} \geq \mathbf{0}_{K,1}$$

Primal variable $\dot{\tilde{\mathbf{q}}}$, which is the time derivative of $\tilde{\mathbf{q}}(t)$, and dual variables $\dot{\boldsymbol{\xi}}$ and $\dot{\boldsymbol{\omega}}$, which are the time derivatives of $\boldsymbol{\xi}(t)$ and $\boldsymbol{\omega}(t)$, may differ in their sign constraints within this family of linear programs and their duals.

Pullan [1995] also showed that as long as $\boldsymbol{\alpha}$, $\mathbf{a}$, and $\mathbf{m}$ are piecewise constant over time with a finite number of discontinuous breakpoints $t_0 \leq t_1 \leq t_2 \leq \cdots \leq t_N = t_0 + T < \infty$, then there is an optimal solution to the SCLP and DCLP with no more than 2^{2K} additional breakpoints where $\dot{\tilde{\mathbf{u}}}^*(t)$, $\dot{\tilde{\mathbf{y}}}^*(t)$, $\boldsymbol{\pi}^*(t)$, and $\boldsymbol{\xi}^*(t)$ are also piecewise constant over time while $\tilde{\mathbf{q}}^*(t)$ and $\boldsymbol{\omega}^*(t)$ are continuous and piecewise affine linear over time. It has also been shown [Weiss 2001], that if SCLP is feasible and bounded, then there exists an optimal solution with piecewise constant $\dot{\tilde{\mathbf{u}}}^*(t)$ and discontinuities of the following form at $t_0 \leq t_1 \leq t_2 \leq \cdots \leq t_N = t_0 + T < \infty$. Let $\dot{\tilde{\mathbf{q}}}$, $\dot{\boldsymbol{\pi}}$, $\dot{\boldsymbol{\xi}}$, and $\dot{\boldsymbol{\omega}}$ be the piecewise-constant time derivatives of $\tilde{\mathbf{q}}(t)$, $\boldsymbol{\pi}(t)$, $\boldsymbol{\xi}(t)$, and $\boldsymbol{\omega}(t)$, respectively. Then in each interval (t_{n-1}, t_n), the values $\dot{\tilde{\mathbf{u}}}$, $\dot{\tilde{\mathbf{y}}}$, $\dot{\tilde{\mathbf{q}}}$, $\dot{\boldsymbol{\pi}}$, $\dot{\boldsymbol{\xi}}$, and $\dot{\boldsymbol{\omega}}$ are complementary slack basic solutions of the primal and dual linear programs given above that are optimal for the following sign constraints:

$$\dot{\tilde{q}}_{j,k} \begin{cases} \geq 0 & \text{if } \tilde{q}_{j,k}(t_{n-1}) = 0 \\ \text{unrestricted} & \text{if } \tilde{q}_{j,k}(t_{n-1}) > 0 \end{cases},$$

$$\dot{\xi}_i \begin{cases} \geq 0 & \text{if } \xi_i(t_n) = 0 \\ \text{unrestricted} & \text{if } \xi_i(t_n) > 0 \end{cases},$$

and

$$\dot{\omega}_{j,k} \begin{cases} \geq 0 & \text{if } \omega_{j,k}(t_n) = 0 \\ \text{unrestricted} & \text{if } \omega_{j,k}(t_n) > 0 \end{cases}.$$

In fact, these basic solutions in neighboring intervals differ only by a single pivot, and the breakpoint time t_n is determined by the pivot, through the constraints:

$\tilde{q}_{j,k}(t_n) = 0$ if $\dot{\tilde{q}}_{j,k}$ leaves the basis at time t_n,

$\xi_i(t_n) = 0$ if $\dot{\tilde{y}}_i$ leaves the basis at time t_n,

and

$\omega_{j,k}(t_n) = 0$ if $\dot{\tilde{u}}_{j,k}$ leaves the basis at time t_n,

A simplex-like algorithm has also been developed by Weiss [2001] to solve an SCLP (using a methodology suggested by these ideas) that incrementally constructs such a optimal piecewise-linear solution for each value of $T \geq t_0$.

5.1 EXAMPLE NETWORK: OPTIMAL SCLP SOLUTION FOR HOLDING COST

An SCLP solution (given by Weiss' simplex-like algorithm) that attains the optimal fluid holding cost objective is plotted in the following figures for the example network. Figure 5.1 shows the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ over time.

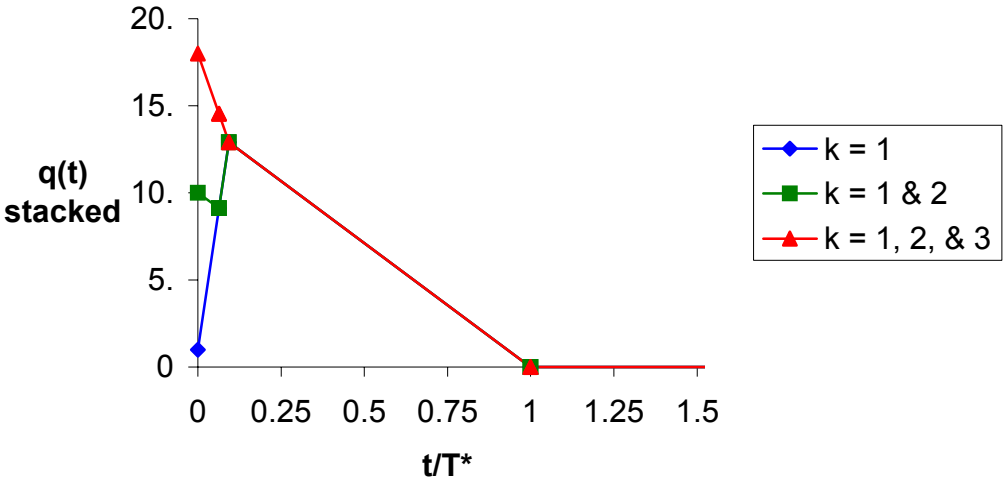

Figure 5.1: Total WIP for Optimal Makespan in Example Network

Figure 5.2 shows the different elements of $\tilde{v}^*_{j,k}(t)$ stacked upon each other (for each machine type) to show what fraction of its capacity the machine type devotes to each queue over time. Figure 5.3 shows $\tilde{u}^*_{j,k}(t)$, the cumulative number of units that have left each queue over time.

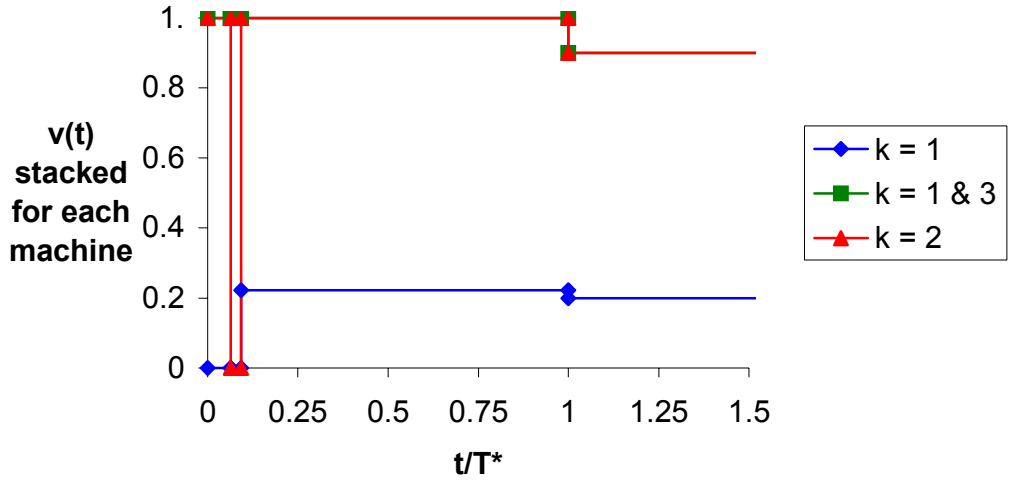

Figure 5.2: Machine Utilization for Optimal Makespan in Example Network

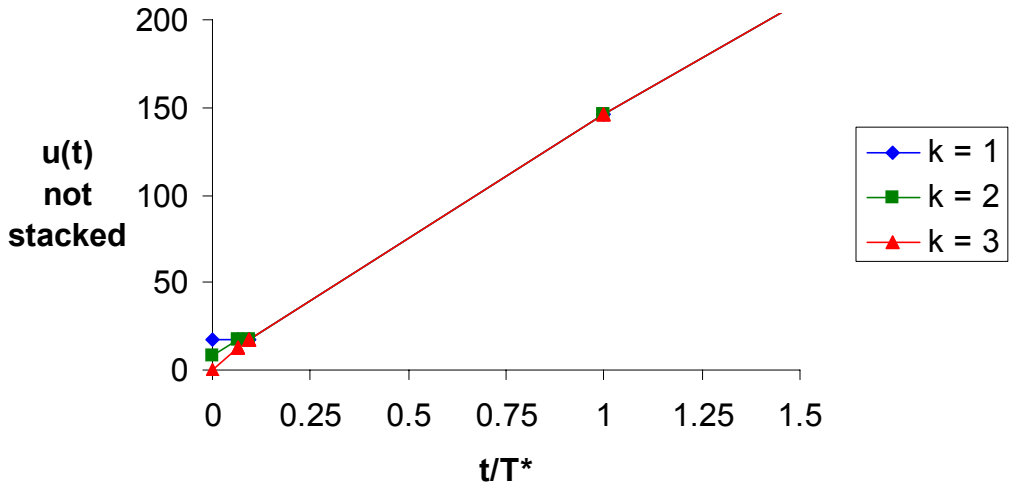

Figure 5.3: Cumulative Units Processed for Optimal Makespan in Example Network

6. Quadratic Programming (QP) Models: Fluid Relaxation

In this section, we model the fluid relaxation as a quadratic program (similar to those proposed by Pullan [1993, 1995, and 1996]), which has the form:

$$\min_{\mathbf{x}} \left(\frac{\mathbf{x}^{\mathrm{T}}\mathbf{H}\mathbf{x}}{2} + \mathbf{f}^{\mathrm{T}}\mathbf{x} \right) \tag{6.1}$$

$$\text{s.t.} \quad \bar{\bar{\mathbf{A}}}\,\mathbf{x} = \bar{\bar{\mathbf{b}}} \tag{6.2}$$

$$\mathbf{A}\mathbf{x} \le \mathbf{b} \tag{6.3}$$

$$\bar{\mathbf{x}} \ge \mathbf{0} \tag{6.4}$$

where the constraints are linear and the objective function is quadratic in $\mathbf{x}$ (the vector of decision variables) and $\bar{\mathbf{x}}$ is the subvector of $\mathbf{x}$ that is constrained to be nonnegative. Input data for the quadratic program comes in the form of matrices $\mathbf{H}$, $\mathbf{A}$, and $\bar{\bar{\mathbf{A}}}$, as well as vectors $\mathbf{b}$ and $\bar{\bar{\mathbf{b}}}$. Several QP formulations are presented and compared for computational efficiency (which is why we differentiate between equality and inequality constraints).

6.1 Quadratic Program (QP) for Fluid Relaxation with Δt, $\tilde{Q}$, and $\Delta\tilde{U}$ as Decision Variables

Since the SCLP formulation (with the weighted holding cost objective function only) has an optimal solution where $\dot{\tilde{\mathbf{u}}}(t)$ is piecewise constant and $\tilde{\mathbf{q}}(t)$ is piecewise linear over time with a finite number of breakpoints (if $\boldsymbol{\alpha}$, $\mathbf{a}$, and $\mathbf{m}$ are similarly constrained), then the equivalent CLP formulation must also have an optimal solution where $\tilde{\mathbf{u}}(t)$ and $\tilde{\mathbf{q}}(t)$ are piecewise linear over time with a finite

number of breakpoints. This suggests a formulation that assumes N breakpoints $t_0 \le t_1 \le t_2 \le \cdots \le t_N = t_0 + T < \infty$ where the decision variables are the queue levels $\tilde{q}_{j,k}(t_n)$, the length of the time intervals $\Delta t_n \equiv t_n - t_{n-1}$, and the increase in the number of units processed $\Delta\tilde{u}_{j,k}(t_n) \equiv \tilde{u}_{j,k}(t_n) - \tilde{u}_{j,k}(t_{n-1})$. The time intervals must be nonnegative,

$$\Delta t_n \ge 0 \qquad \forall n \in \{1,2,...,N\},$$

and in fact must be strictly positive if N is the minimal number of breakpoints required to give an optimal solution to the SCLP formulation.

Initially,

$$\tilde{q}_{j,k}(t_0) = q_{j,k}(t_0) \qquad \begin{cases} \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases},$$

and, if the stability condition $\rho_i < 1$ is met $\forall i \in \{1,2,...,I\}$, we know that for large enough N,

$$\tilde{q}_{j,k}(t) = 0 \qquad \begin{cases} \forall t \ge t_N \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}.$$

Using the trapezoid rule (exact since $\tilde{\mathbf{q}}(t)$ is piecewise linear), the weighted holding cost objective function (4.16) becomes (for $T \ge \sum_{n=1}^{N} \Delta t_n$):

$$\tilde{C}_{\mathrm{h}}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}} \mid T, ...) = \int_{t_0}^{t_0+T} \mathbf{c}^{\mathrm{T}} \tilde{\mathbf{q}}(t) dt$$

$$= \mathbf{c}^{\mathrm{T}} \int_{t_0}^{t_0+T} \tilde{\mathbf{q}}(t) dt$$

$$= \mathbf{c}^{\mathrm{T}} \sum_{n=1}^{N} \frac{\tilde{\mathbf{q}}(t_{n-1}) + \tilde{\mathbf{q}}(t_n)}{2} \Delta t_n$$

$$= \frac{\mathbf{c}^{\mathrm{T}}}{2} \left[\tilde{\mathbf{q}}(t_0)\Delta t_1 + \sum_{n=1}^{N-1} \tilde{\mathbf{q}}(t_n)\left(\Delta t_n + \Delta t_{n+1}\right) + \tilde{\mathbf{q}}(t_N)\Delta t_N \right]$$

$$= \frac{\mathbf{c}^{\mathrm{T}}}{2} \left[\mathbf{q}(t_0)\Delta t_1 + \sum_{n=1}^{N-1} \tilde{\mathbf{q}}(t_n)\left(\Delta t_n + \Delta t_{n+1}\right) \right]$$

where we have separated the linear term from the quadratic terms. Thus, we can create a new objective function that is equivalent to the weighted holding cost objective:

$$\min_{\Delta\mathbf{t},\tilde{\mathbf{Q}},\Delta\tilde{\mathbf{U}}} \tilde{C}_{\mathrm{h}}(\Delta\mathbf{t}, \tilde{\mathbf{Q}}, \Delta\tilde{\mathbf{U}} \mid N, ...) = \frac{\mathbf{c}^{\mathrm{T}}}{2} \left[\mathbf{q}(t_0)\Delta t_1 + \sum_{n=1}^{N-1} \tilde{\mathbf{q}}(t_n)\left(\Delta t_n + \Delta t_{n+1}\right) \right]$$

where

$$\Delta\mathbf{t} \equiv \begin{bmatrix} \Delta t_1 & \Delta t_2 & \cdots & \Delta t_N \end{bmatrix}^{\mathrm{T}},$$

$$\tilde{\mathbf{Q}} \equiv \begin{bmatrix} \tilde{\mathbf{q}}(t_1) & \tilde{\mathbf{q}}(t_2) & \cdots & \tilde{\mathbf{q}}(t_{N-1}) \end{bmatrix},$$

and

$$\Delta\tilde{\mathbf{U}} \equiv \begin{bmatrix} \Delta\tilde{\mathbf{u}}(t_1) & \Delta\tilde{\mathbf{u}}(t_2) & \cdots & \Delta\tilde{\mathbf{u}}(t_N) \end{bmatrix}.$$

Constraints (4.20), (4.17), and (4.18) become:

$$\tilde{\mathbf{q}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{2, 3, ..., N-1\},$$

$$\Delta\tilde{\mathbf{u}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1, 2, ..., N\},$$

and

$$\mathbf{B}\mathbf{D}(\mathbf{p})\Delta\tilde{\mathbf{u}}(t_n) - \mathbf{D}(\mathbf{a})\mathbf{m}\Delta t_n + \Delta\tilde{\mathbf{y}}(t_n) = \mathbf{0}_{I,1} \qquad \forall n \in \{1, 2, ..., N\}. \tag{6.5}$$

Here again, we have inserted a slack variable $\Delta\tilde{\mathbf{y}}(t_n)$ into constraint (6.5), so

$$\Delta\tilde{\mathbf{y}}(t_n) \geq \mathbf{0}_{I,1} \qquad \forall t \geq t_0 .$$

The (possibly non-integer) number of idle machines in the *n*th time period is then given by

$$\dot{\tilde{\mathbf{y}}}(t_n) = \frac{\Delta\tilde{\mathbf{y}}(t_n)}{\Delta t_n} .$$

Constraint (4.21) becomes the three constraints:

$$\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\Delta\tilde{\mathbf{u}}(t_1) + \tilde{\mathbf{q}}(t_1) - \boldsymbol{\alpha}\Delta t_1 = \mathbf{q}(t_0), \tag{6.6}$$

$$\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\Delta\tilde{\mathbf{u}}(t_n) + \tilde{\mathbf{q}}(t_n) - \tilde{\mathbf{q}}(t_{n-1}) - \boldsymbol{\alpha}\Delta t_n = \mathbf{0}_{K,1} \quad \forall n \in \{2,3,...,N-1\}, \tag{6.7}$$

and

$$\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\Delta\tilde{\mathbf{u}}(t_N) - \tilde{\mathbf{q}}(t_{N-1}) - \boldsymbol{\alpha}\Delta t_N = \mathbf{0}_{K,1} . \tag{6.8}$$

Putting all these equations together gives Formulation 6.1, a QP with $\Delta\mathbf{t}$, $\tilde{\mathbf{Q}}$, and (although not in the objective function) $\Delta\tilde{\mathbf{U}}$ as the primary decision variables and with $\Delta\tilde{\mathbf{Y}} \equiv \left[\Delta\tilde{\mathbf{y}}(t_1) \quad \Delta\tilde{\mathbf{y}}(t_2) \quad \cdots \quad \Delta\tilde{\mathbf{y}}(t_N)\right]$ containing slack variables for the inequality constraints (so they appear as objective function arguments but not under the "min").

Formulation 6.1: Quadratic Program (QP) for Fluid Relaxation with $\Delta\mathbf{t}$, $\tilde{\mathbf{Q}}$, and $\Delta\tilde{\mathbf{U}}$ as Decision Variables

$$\min_{\Delta\mathbf{t},\tilde{\mathbf{Q}},\Delta\tilde{\mathbf{U}}} \tilde{C}_{\mathrm{h}}(\Delta\mathbf{t},\tilde{\mathbf{Q}},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}} \mid N,...) = \frac{\mathbf{c}^{\mathrm{T}}}{2}\left[\mathbf{q}(t_0)\Delta t_1 + \sum_{n=1}^{N-1}\tilde{\mathbf{q}}(t_n)\left(\Delta t_n + \Delta t_{n+1}\right)\right]$$

s.t.
$$\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\Delta\tilde{\mathbf{u}}(t_1) + \tilde{\mathbf{q}}(t_1) - \boldsymbol{\alpha}\Delta t_1 = \mathbf{q}(t_0)$$
$$\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\Delta\tilde{\mathbf{u}}(t_n) + \tilde{\mathbf{q}}(t_n) - \tilde{\mathbf{q}}(t_{n-1}) - \boldsymbol{\alpha}\Delta t_n = \mathbf{0}_{K,1} \quad \forall n \in \{2,3,...,N-1\}$$
$$\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\Delta\tilde{\mathbf{u}}(t_N) - \tilde{\mathbf{q}}(t_{N-1}) - \boldsymbol{\alpha}\Delta t_N = \mathbf{0}_{K,1}$$
$$\mathbf{BD}(\mathbf{p})\Delta\tilde{\mathbf{u}}(t_n) - \mathbf{D}(\mathbf{a})\mathbf{m}\Delta t_n + \Delta\tilde{\mathbf{y}}(t_n) = \mathbf{0}_{I,1} \quad \forall n \in \{1,2,...,N\}$$
$$\Delta t_n \geq 0 \quad \forall n \in \{1,2,...,N\}$$
$$\tilde{\mathbf{q}}(t_n) \geq \mathbf{0}_{K,1} \quad \forall n \in \{1,2,...,N-1\}$$
$$\Delta\tilde{\mathbf{u}}(t_n) \geq \mathbf{0}_{K,1} \quad \forall n \in \{1,2,...,N\}$$
$$\Delta\tilde{\mathbf{y}}(t_n) \geq \mathbf{0}_{I,1} \quad \forall n \in \{1,2,...,N\}$$

Since the constraints are linear and the objective function is quadratic in the decision variables, this is a quadratic program of the form given in equations (6.1) through (6.4) where the slack variables are implicit in the inequalities. The vectors and matrices in equations (6.1) through (6.4) have the following form:

$$\mathbf{x} = \overset{\geq}{\mathbf{x}} = \begin{bmatrix} \Delta\mathbf{t} \\ \mathbf{x}_{\Delta\mathbf{U}} \\ \mathbf{x}_{\mathbf{Q}} \end{bmatrix},\ \mathbf{f} = \begin{bmatrix} \mathbf{f}_{\Delta\mathbf{t}} \\ \mathbf{0}_{NK,1} \\ \mathbf{0}_{(N-1)K,1} \end{bmatrix},\ \mathbf{H} = \begin{bmatrix} \mathbf{0}_{N,N} & \mathbf{0}_{N,NK} & \mathbf{H}_{\mathbf{Q},\Delta\mathbf{t}}^{\mathrm{T}} \\ \mathbf{0}_{NK,N} & \mathbf{0}_{NK,NK} & \mathbf{0}_{NK,(N-1)K} \\ \mathbf{H}_{\mathbf{Q},\Delta\mathbf{t}} & \mathbf{0}_{(N-1)K,NK} & \mathbf{0}_{(N-1)K,(N-1)K} \end{bmatrix},$$

$$\overset{=}{\mathbf{A}} = \begin{bmatrix} \overset{=}{\mathbf{A}}_{\Delta\mathbf{t}} & \overset{=}{\mathbf{A}}_{\Delta\mathbf{U}} & \overset{=}{\mathbf{A}}_{\mathbf{Q}} \end{bmatrix},\ \mathbf{A} = \begin{bmatrix} \mathbf{A}_{\Delta\mathbf{t}} & \mathbf{A}_{\Delta\mathbf{U}} & \mathbf{0}_{NI,(N-1)K} \end{bmatrix}, \text{ and } \mathbf{b} = \mathbf{0}_{NI,1},$$

where the submatrices are as follows (with blank subsubmatrices indicating zeros and with lines separating subsubmatrices corresponding to different values of *n*):

$$\mathbf{x}_{\Delta\mathbf{U}} = \begin{bmatrix} \Delta\tilde{\mathbf{u}}(t_1) \\ \hline \Delta\tilde{\mathbf{u}}(t_2) \\ \hline \vdots \\ \hline \Delta\tilde{\mathbf{u}}(t_N) \end{bmatrix}, \quad \mathbf{x}_{\mathbf{Q}} = \begin{bmatrix} \tilde{\mathbf{q}}(t_1) \\ \hline \tilde{\mathbf{q}}(t_2) \\ \hline \vdots \\ \hline \tilde{\mathbf{q}}(t_{N-1}) \end{bmatrix}, \quad \mathbf{f}_{\Delta\mathbf{t}} = \begin{bmatrix} \mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)/2 \\ \hline \\ \hline \vdots \\ \hline \\ \end{bmatrix},$$

$$\mathbf{H}_{\mathbf{Q},\Delta\mathbf{t}} = \frac{1}{2}\begin{bmatrix} \mathbf{c} & \mathbf{c} & & & \\ \hline & \mathbf{c} & \mathbf{c} & & \\ \hline & & \ddots & \ddots & \\ \hline & & & \mathbf{c} & \mathbf{c} \end{bmatrix}, \quad \bar{\mathbf{b}} = \begin{bmatrix} \mathbf{q}(t_0) \\ \hline \\ \hline \\ \hline \vdots \\ \hline \\ \end{bmatrix}, \quad \bar{\mathbf{A}}_{\Delta\mathbf{t}} = \begin{bmatrix} -\boldsymbol{\alpha} & & & \\ \hline & -\boldsymbol{\alpha} & & \\ \hline & & \ddots & \\ \hline & & & -\boldsymbol{\alpha} \end{bmatrix},$$

$$\bar{\mathbf{A}}_{\Delta\mathbf{U}} = \begin{bmatrix} \mathbf{I}_K - \mathbf{P}^{\mathrm{T}} & & & \\ \hline & \mathbf{I}_K - \mathbf{P}^{\mathrm{T}} & & \\ \hline & & \ddots & \\ \hline & & & \mathbf{I}_K - \mathbf{P}^{\mathrm{T}} \end{bmatrix}, \quad \bar{\mathbf{A}}_{\mathbf{Q}} = \begin{bmatrix} \mathbf{I}_K & & & \\ \hline -\mathbf{I}_K & \mathbf{I}_K & & \\ \hline & -\mathbf{I}_K & \ddots & \\ \hline & & \ddots & \mathbf{I}_K \\ \hline & & & -\mathbf{I}_K \end{bmatrix},$$

$$\mathbf{A}_{\Delta\mathbf{t}} = \begin{bmatrix} -\mathbf{D}(\mathbf{a})\mathbf{m} & & & \\ \hline & -\mathbf{D}(\mathbf{a})\mathbf{m} & & \\ \hline & & \ddots & \\ \hline & & & -\mathbf{D}(\mathbf{a})\mathbf{m} \end{bmatrix}, \text{ and}$$

$$\mathbf{A}_{\Delta\mathbf{U}} = \begin{bmatrix} \mathbf{BD}(\mathbf{p}) & & & \\ \hline & \mathbf{BD}(\mathbf{p}) & & \\ \hline & & \ddots & \\ \hline & & & \mathbf{BD}(\mathbf{p}) \end{bmatrix}.$$

These vectors and matrices have the dimensions and sparsity shown in Table 6.1:

Table 6.1: Dimensions and Sparsity of Vectors and Matrices in QP Formulation with Δt, $\tilde{Q}$, and $\Delta\tilde{U}$ as Decision Variables

Name	*Number of Rows/Columns*	*Number of Elements*	*Maximum Number of Nonzero Elements*
x	$N(2K+1)-K$ / 1	-	-
H	$N(2K+1)-K$ / $N(2K+1)-K$	$N^2(4K^2+4K+1)$ $+N(-4K^2-2K)+K^2$	$4NK$ $-4K$
f	$N(2K+1)-K$ / 1	$N(2K+1)$ $-K$	1
A and $\underline{=}$ **A**	$N(K+I)$ / $N(2K+1)-K$	$N^2(2K^2+2IK+K+I)$ $+N(-K^2-IK)$	$N(K^2+4K+I)$ $-2K$ or, if **P** is simple, $N(6K+I-J)$ $-2K$ or, if **α** is too, $N(5K+I)$ $-2K$
b and $\underline{=}$ **b**	$N(K+I)$ / 1	$N(K+I)$	K
Total	-	$N^2(6K^2+2IK+5K+I+1)$ $+N(-5K^2-IK+K+I+1)$ $+K^2-K$	$N(K^2+8K+I)$ $-5K+1$ or, if **P** is simple, $N(10K+I-J)$ $-5K+1$ or, if **α** is too, $N(9K+I)$ $-5K+1$

Here we say the vector **α** of arrival rates is simple if it has only J nonzero entries (one for each job type). We say the routing matrix **P** is *simple* if it consists of only zeros with some ones on the first superdiagonal.

6.2 QUADRATIC PROGRAM (QP) FOR FLUID RELAXATION WITH Δt AND $\tilde{Q}$ AS DECISION VARIABLES

In general, $\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)$ is easily inverted, so we can solve for $\Delta\tilde{\mathbf{u}}(t_n)$ in constraints (6.6), (6.7), and (6.8) to get the following definitions for $\Delta\tilde{\mathbf{u}}(t_n)$:

$$\Delta\tilde{\mathbf{u}}(t_1) = \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_1 + \mathbf{q}(t_0) - \tilde{\mathbf{q}}(t_1)\right],$$

$$\Delta\tilde{\mathbf{u}}(t_n) = \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_n + \tilde{\mathbf{q}}(t_{n-1}) - \tilde{\mathbf{q}}(t_n)\right] \qquad \forall n \in \{2,3,...,N-1\},$$

and

$$\Delta\tilde{\mathbf{u}}(t_N) = \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_N + \tilde{\mathbf{q}}(t_{N-1})\right].$$

Recasting these definitions for $\Delta\tilde{\mathbf{u}}(t_n)$ as constraints gives the following:

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_1 - \tilde{\mathbf{q}}(t_1)\right] + \Delta\tilde{\mathbf{u}}(t_1) = \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\mathbf{q}(t_0),$$

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_n + \tilde{\mathbf{q}}(t_{n-1}) - \tilde{\mathbf{q}}(t_n)\right] + \Delta\tilde{\mathbf{u}}(t_n) = \mathbf{0}_{K,1} \qquad \forall n \in \{2,3,...,N-1\},$$

and

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_N + \tilde{\mathbf{q}}(t_{N-1})\right] + \Delta\tilde{\mathbf{u}}(t_N) = \mathbf{0}_{K,1}.$$

Plugging the definitions for $\Delta\tilde{\mathbf{u}}(t_n)$ into constraint (6.5) gives the following constraints:

$$-\boldsymbol{\chi}\Delta t_1 - \mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\tilde{\mathbf{q}}(t_1) + \Delta\tilde{\mathbf{y}}(t_1) = -\mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\mathbf{q}(t_0),$$

$$-\boldsymbol{\chi}\Delta t_n + \mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\tilde{\mathbf{q}}(t_{n-1}) - \tilde{\mathbf{q}}(t_n)\right] + \Delta\tilde{\mathbf{y}}(t_n) = \mathbf{0}_{I,1} \qquad \forall n \in \{2,3,...,N-1\},$$

and

$$-\boldsymbol{\chi}\Delta t_N + \mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\tilde{\mathbf{q}}(t_{N-1}) + \Delta\tilde{\mathbf{y}}(t_N) = \mathbf{0}_{I,1},$$

where we have substituted in the vector of discretionary machine availabilities:

$\boldsymbol{\chi} \equiv \mathbf{D}(\mathbf{a})\mathbf{m} - \mathbf{B}\mathbf{D}(\mathbf{p})\boldsymbol{\alpha}^{+}$.

Putting all the remaining equations together gives Formulation 6.2, a QP with $\Delta\mathbf{t}$ and $\tilde{\mathbf{Q}}$ as the primary decision variables and with $\Delta\tilde{\mathbf{U}}$ and $\Delta\tilde{\mathbf{Y}}$ as slack variables for their respective constraints (so they appear as objective function arguments but not under the "min").

Formulation 6.2: Quadratic Program (QP) for Fluid Relaxation with $\Delta\mathbf{t}$ and $\tilde{\mathbf{Q}}$ as Decision Variables

$$\min_{\Delta\mathbf{t},\tilde{\mathbf{Q}}} \tilde{C}_{\mathrm{h}}\left(\Delta\mathbf{t},\tilde{\mathbf{Q}},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}} \mid N,...\right) = \frac{\mathbf{c}^{\mathrm{T}}}{2}\left[\mathbf{q}(t_0)\Delta t_1 + \sum_{n=1}^{N-1}\tilde{\mathbf{q}}(t_n)\left(\Delta t_n + \Delta t_{n+1}\right)\right]$$

s.t.

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_1 - \tilde{\mathbf{q}}(t_1)\right] + \Delta\tilde{\mathbf{u}}(t_1) = \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\mathbf{q}(t_0)$$

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_n + \tilde{\mathbf{q}}(t_{n-1}) - \tilde{\mathbf{q}}(t_n)\right] + \Delta\tilde{\mathbf{u}}(t_n) = \mathbf{0}_{K,1} \qquad \forall n \in \{2,3,...,N-1\}$$

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\boldsymbol{\alpha}\Delta t_N + \tilde{\mathbf{q}}(t_{N-1})\right] + \Delta\tilde{\mathbf{u}}(t_N) = \mathbf{0}_{K,1}$$

$$-\boldsymbol{\chi}\Delta t_1 - \mathbf{B}\mathbf{D}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\tilde{\mathbf{q}}(t_1) + \Delta\tilde{\mathbf{y}}(t_1) = -\mathbf{B}\mathbf{D}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\mathbf{q}(t_0)$$

$$-\boldsymbol{\chi}\Delta t_n + \mathbf{B}\mathbf{D}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\left[\tilde{\mathbf{q}}(t_{n-1}) - \tilde{\mathbf{q}}(t_n)\right] + \Delta\tilde{\mathbf{y}}(t_n) = \mathbf{0}_{I,1} \qquad \forall n \in \{2,3,...,N-1\}$$

$$-\boldsymbol{\chi}\Delta t_N + \mathbf{B}\mathbf{D}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}\tilde{\mathbf{q}}(t_{N-1}) + \Delta\tilde{\mathbf{y}}(t_N) = \mathbf{0}_{I,1}$$

$$\Delta t_n \geq 0 \qquad \forall n \in \{1,2,...,N\}$$

$$\tilde{\mathbf{q}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,...,N-1\}$$

$$\Delta\tilde{\mathbf{u}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,...,N\}$$

$$\Delta\tilde{\mathbf{y}}(t_n) \geq \mathbf{0}_{I,1} \qquad \forall n \in \{1,2,...,N\}$$

Equation (6.2) is no longer needed since there are no equality constraints. The vectors and matrices in equations (6.1), (6.3), and (6.4) now have the following form:

$$\mathbf{x} = \overset{\geq}{\mathbf{x}} = \begin{bmatrix} \Delta\mathbf{t} \\ \mathbf{x}_{\mathbf{Q}} \end{bmatrix},\ \mathbf{f} = \begin{bmatrix} \mathbf{f}_{\Delta\mathbf{t}} \\ \mathbf{0}_{(N-1)K,1} \end{bmatrix},\ \mathbf{H} = \begin{bmatrix} \mathbf{0}_{N,N} & \mathbf{H}_{\mathbf{Q},\Delta\mathbf{t}}^{\mathrm{T}} \\ \mathbf{H}_{\mathbf{Q},\Delta\mathbf{t}} & \mathbf{0}_{(N-1)K,(N-1)K} \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{\Delta\mathbf{U},\Delta\mathbf{t}} & \mathbf{A}_{\Delta\mathbf{U},\mathbf{Q}} \\ \mathbf{A}_{\Delta\mathbf{Y},\Delta\mathbf{t}} & \mathbf{A}_{\Delta\mathbf{Y},\mathbf{Q}} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{b}_{\Delta\mathbf{U}} \\ \mathbf{b}_{\Delta\mathbf{Y}} \end{bmatrix},$$

where the submatrices are as follows (with blank subsubmatrices indicating zeros and with lines separating subsubmatrices corresponding to different values of n):

$$\mathbf{x}_{\mathbf{Q}} = \begin{bmatrix} \tilde{\mathbf{q}}(t_1) \\ \hline \tilde{\mathbf{q}}(t_2) \\ \hline \vdots \\ \hline \tilde{\mathbf{q}}(t_{N-1}) \end{bmatrix},\ \mathbf{f}_{\Delta\mathbf{t}} = \begin{bmatrix} \mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)/2 \\ \hline \\ \hline \vdots \\ \hline \\ \hline \end{bmatrix},\ \mathbf{H}_{\mathbf{Q},\Delta\mathbf{t}} = \frac{1}{2}\left[\begin{array}{c|c|c|c|c} \mathbf{c} & \mathbf{c} & & & \\ \hline & \mathbf{c} & \mathbf{c} & & \\ \hline & & \ddots & \ddots & \\ \hline & & & \mathbf{c} & \mathbf{c} \end{array}\right],$$

$$\mathbf{A}_{\Delta\mathbf{U},\Delta\mathbf{t}} = \left[\begin{array}{c|c|c|c} -\boldsymbol{\alpha}^{+} & & & \\ \hline & -\boldsymbol{\alpha}^{+} & & \\ \hline & & \ddots & \\ \hline & & & -\boldsymbol{\alpha}^{+} \end{array}\right],\ \mathbf{A}_{\Delta\mathbf{Y},\Delta\mathbf{t}} = \left[\begin{array}{c|c|c|c} -\boldsymbol{\chi} & & & \\ \hline & -\boldsymbol{\chi} & & \\ \hline & & \ddots & \\ \hline & & & -\boldsymbol{\chi} \end{array}\right],$$

$$\mathbf{A}_{\Delta\mathbf{U},\mathbf{Q}} = \left[\begin{array}{c|c|c|c} \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} & & & \\ \hline -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} & \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} & & \\ \hline & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} & \ddots & \\ \hline & & \ddots & \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \\ \hline & & & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \end{array}\right],$$

$$\mathbf{A}_{\Delta\mathbf{Y},\mathbf{Q}} = \left[\begin{array}{c|c|c|c} -\mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} & & & \\ \hline \mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} & -\mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} & & \\ \hline & \mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} & \ddots & \\ \hline & & \ddots & -\mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \\ \hline & & & \mathbf{BD}(\mathbf{p})\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \end{array}\right],$$

$$\mathbf{b}_{\Delta\mathbf{U}} = \left[\begin{array}{c} \mathbf{q}^{+}(t_0) \\ \hline \vdots \\ \hline \end{array}\right], \text{ and } \mathbf{b}_{\Delta\mathbf{Y}} = \left[\begin{array}{c} -\mathbf{BD}(\mathbf{p})\mathbf{q}^{+}(t_0) \\ \hline \vdots \\ \hline \end{array}\right].$$

These vectors and matrices have the dimensions and sparsity shown in Table 6.2:

Table 6.2: Dimensions and Sparsity of Vectors and Matrices in QP Formulation with Δt and $\tilde{\mathbf{Q}}$ as Decision Variables

Name	*Number of Rows/Columns*	*Number of Elements*	*Maximum Number of Nonzero Elements*
x	$N(K+1)-K$ / 1	-	-
H	$N(K+1)-K$ / $N(K+1)-K$	$N^2(K^2+2K+1)$ $+N(-2K^2-2K)+K^2$	$4NK$ $-4K$
f	$N(K+1)-K$ / 1	$N(K+1)$ $-K$	1
A	$N(K+I)$ / $N(K+1)-K$	$N^2(K^2+IK+K+I)$ $+N(-K^2-IK)$	$N(2K^2+2IK+K+I)$ $-2K^2-2IK$
b	$N(K+I)$ / 1	$N(K+I)$	$K+I$
Total	-	$N^2(2K^2+IK+3K+I+1)$ $+N(-3K^2-IK+I+1)$ $+K^2-K$	$N(2K^2+2IK+5K+I)$ $-2K^2-2IK-3K+I$

6.3 QUADRATIC PROGRAM (QP) FOR FLUID RELAXATION WITH $\Delta\mathbf{t}$ AND $\underline{\tilde{\mathbf{Q}}}$ AS DECISION VARIABLES

In order to avoid explicitly computing $\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}$ (which may be computationally expensive and prone to numerical error), we can make the following substitutions into the last QP formulation: the total (immediate and upstream) WIP

$$\mathbf{q}^{+}(t) \equiv \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \mathbf{q}(t),$$

the upstream arrival rates

$$\boldsymbol{\alpha}^{+} \equiv \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \boldsymbol{\alpha},$$

(both which can be computed by Gaussian elimination instead of matrix inversion), and the keeping cost

$$\mathbf{c}^{+} \equiv \left(\mathbf{I}_K - \mathbf{P}\right)\mathbf{c}.$$

If $\mathbf{P}$ is simple and $\mathbf{c}$ is constant over the stages of each product type, then $\mathbf{c}^{+}$ has only J nonzeros.

Making these substitutions gives Formulation 6.3, a QP with $\Delta\mathbf{t}$ and $\tilde{\mathbf{Q}}^{+} \equiv \left[\tilde{\mathbf{q}}^{+}(t_1) \quad \tilde{\mathbf{q}}^{+}(t_2) \quad \cdots \quad \tilde{\mathbf{q}}^{+}(t_{N-1})\right]$ as the primary decision variables and with $\Delta\tilde{\mathbf{U}}$, $\Delta\tilde{\mathbf{Y}}$, and $\tilde{\mathbf{Q}}$ as slack variables for their respective constraints (so they appear as objective function arguments but not under the "min").

Formulation 6.3: Quadratic Program (QP) for Fluid Relaxation with $\Delta\mathbf{t}$ and $\tilde{\mathbf{Q}}^{+}$ as Decision Variables

$$\min_{\Delta\mathbf{t},\tilde{\mathbf{Q}}^{+}} \tilde{C}_{\mathrm{h}}\left(\Delta\mathbf{t},\tilde{\mathbf{Q}}^{+},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}},\tilde{\mathbf{Q}} \mid N,\ldots\right) = \frac{\left(\mathbf{c}^{+}\right)^{\mathrm{T}}}{2}\left[\mathbf{q}^{+}(t_0)\Delta t_1 + \sum_{n=1}^{N-1}\tilde{\mathbf{q}}^{+}(t_n)\left(\Delta t_n + \Delta t_{n+1}\right)\right]$$

s.t.

$$-\boldsymbol{\alpha}^{+}\Delta t_1 + \tilde{\mathbf{q}}^{+}(t_1) + \Delta\tilde{\mathbf{u}}(t_1) = \mathbf{q}^{+}(t_0)$$

$$-\boldsymbol{\alpha}^{+}\Delta t_n - \tilde{\mathbf{q}}^{+}(t_{n-1}) + \tilde{\mathbf{q}}^{+}(t_n) + \Delta\tilde{\mathbf{u}}(t_n) = \mathbf{0}_{K,1} \qquad \forall n \in \{2,3,\ldots,N-1\}$$

$$-\boldsymbol{\alpha}^{+}\Delta t_N - \tilde{\mathbf{q}}^{+}(t_{N-1}) + \Delta\tilde{\mathbf{u}}(t_N) = \mathbf{0}_{K,1}$$

$$-\boldsymbol{\chi}\Delta t_1 - \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_1) + \Delta\tilde{\mathbf{y}}(t_1) = -\mathbf{BD}(\mathbf{p})\mathbf{q}^{+}(t_0)$$

$$-\boldsymbol{\chi}\Delta t_n + \mathbf{BD}(\mathbf{p})\left[\tilde{\mathbf{q}}^{+}(t_{n-1}) - \tilde{\mathbf{q}}^{+}(t_n)\right] + \Delta\tilde{\mathbf{y}}(t_n) = \mathbf{0}_{I,1} \qquad \forall n \in \{2,3,\ldots,N-1\}$$

$$-\boldsymbol{\chi}\Delta t_N + \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{N-1}) + \Delta\tilde{\mathbf{y}}(t_N) = \mathbf{0}_{I,1}$$

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\tilde{\mathbf{q}}^{+}(t_n) + \tilde{\mathbf{q}}(t_n) = \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,\ldots,N-1\}$$

$$\Delta t_n \geq 0 \qquad \forall n \in \{1,2,\ldots,N\}$$

$$\tilde{\mathbf{q}}^{+}(t_n) \in \mathbb{R}^{K}\left(\text{unrestricted}\right) \qquad \forall n \in \{1,2,\ldots,N-1\}$$

$$\tilde{\mathbf{q}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,\ldots,N-1\}$$

$$\Delta\tilde{\mathbf{u}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,\ldots,N\}$$

$$\Delta\tilde{\mathbf{y}}(t_n) \geq \mathbf{0}_{I,1} \qquad \forall n \in \{1,2,\ldots,N\}$$

The vectors and matrices in equations (6.1), (6.3), and (6.4) now have the following form:

$$\mathbf{x} = \begin{bmatrix} \Delta\mathbf{t} \\ \mathbf{x}_{\tilde{\mathbf{Q}}^{+}} \end{bmatrix},\ \overset{\geq}{\mathbf{x}} = \Delta\mathbf{t},\ \mathbf{f} = \begin{bmatrix} \mathbf{f}_{\Delta\mathbf{t}} \\ \mathbf{0}_{(N-1)K,1} \end{bmatrix},\ \mathbf{H} = \begin{bmatrix} \mathbf{0}_{N,N} & \mathbf{H}^{\mathrm{T}}_{\tilde{\mathbf{Q}}^{+},\Delta\mathbf{t}} \\ \mathbf{H}_{\tilde{\mathbf{Q}}^{+},\Delta\mathbf{t}} & \mathbf{0}_{(N-1)K,(N-1)K} \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{\Delta\mathbf{U},\Delta\mathbf{t}} & \mathbf{A}_{\Delta\mathbf{U},\tilde{\mathbf{Q}}^{+}} \\ \mathbf{A}_{\Delta\mathbf{Y},\Delta\mathbf{t}} & \mathbf{A}_{\Delta\mathbf{Y},\tilde{\mathbf{Q}}^{+}} \\ \mathbf{0}_{(N-1)K,N} & \mathbf{A}_{\mathbf{Q},\tilde{\mathbf{Q}}^{+}} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{b}_{\Delta\mathbf{U}} \\ \mathbf{b}_{\Delta\mathbf{Y}} \\ \mathbf{0}_{(N-1)K,1} \end{bmatrix},$$

where the submatrices are as follows (with blank subsubmatrices indicating zeros and with lines separating subsubmatrices corresponding to different values of *n*):

$$\mathbf{x}_{\tilde{\mathbf{Q}}^+} = \begin{bmatrix} \tilde{\mathbf{q}}^+(t_1) \\ \hline \tilde{\mathbf{q}}^+(t_2) \\ \hline \vdots \\ \hline \tilde{\mathbf{q}}^+(t_{N-1}) \end{bmatrix}, \quad \mathbf{f}_{\Delta\mathbf{t}} = \begin{bmatrix} \mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)/2 \\ \hline \\ \hline \vdots \\ \hline \\ \end{bmatrix}, \quad \mathbf{H}_{\tilde{\mathbf{Q}}^+,\Delta\mathbf{t}} = \frac{1}{2}\begin{bmatrix} \mathbf{c}^+ & \mathbf{c}^+ & & & \\ \hline & \mathbf{c}^+ & \mathbf{c}^+ & & \\ \hline & & \ddots & \ddots & \\ \hline & & & \mathbf{c}^+ & \mathbf{c}^+ \end{bmatrix}, \tag{6.9}$$

$$\mathbf{A}_{\Delta\mathbf{U},\Delta\mathbf{t}} = \begin{bmatrix} -\boldsymbol{\alpha}^+ & & & \\ \hline & -\boldsymbol{\alpha}^+ & & \\ \hline & & \ddots & \\ \hline & & & -\boldsymbol{\alpha}^+ \end{bmatrix}, \tag{6.10}$$

$$\mathbf{A}_{\Delta\mathbf{U},\tilde{\mathbf{Q}}^+} = \begin{bmatrix} \mathbf{I}_K & & & \\ \hline -\mathbf{I}_K & \mathbf{I}_K & & \\ \hline & -\mathbf{I}_K & \ddots & \\ \hline & & \ddots & \mathbf{I}_K \\ \hline & & & -\mathbf{I}_K \end{bmatrix}, \quad \mathbf{b}_{\Delta\mathbf{U}} = \begin{bmatrix} \mathbf{q}^+(t_0) \\ \hline \\ \hline \\ \hline \vdots \\ \hline \\ \end{bmatrix} \tag{6.11}$$

$$\mathbf{A}_{\Delta\mathbf{Y},\Delta\mathbf{t}} = \begin{bmatrix} -\boldsymbol{\chi} & & & \\ \hline & -\boldsymbol{\chi} & & \\ \hline & & \ddots & \\ \hline & & & -\boldsymbol{\chi} \end{bmatrix}, \tag{6.12}$$

$$\mathbf{A}_{\Delta\mathbf{Y},\tilde{\mathbf{Q}}^+} = \begin{bmatrix} -\mathbf{BD}(\mathbf{p}) & & & \\ \hline \mathbf{BD}(\mathbf{p}) & -\mathbf{BD}(\mathbf{p}) & & \\ \hline & \mathbf{BD}(\mathbf{p}) & \ddots & \\ \hline & & \ddots & -\mathbf{BD}(\mathbf{p}) \\ \hline & & & \mathbf{BD}(\mathbf{p}) \end{bmatrix}, \quad \mathbf{b}_{\Delta\mathbf{Y}} = \begin{bmatrix} -\mathbf{BD}(\mathbf{p})\mathbf{q}^+(t_0) \\ \hline \\ \hline \\ \hline \vdots \\ \hline \\ \end{bmatrix}, \tag{6.13}$$

and

$$\mathbf{A}_{\mathbf{Q},\tilde{\mathbf{Q}}^+} = \left[\begin{array}{c|c|c|c} -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) & & & \\ \hline & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) & & \\ \hline & & \ddots & \\ \hline & & & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) \end{array}\right]. \tag{6.14}$$

These vectors and matrices have the dimensions and sparsity shown in Table 6.3:

Table 6.3: Dimensions and Sparsity of Vectors and Matrices in QP Formulation with Δt and $\tilde{\mathbf{Q}}^+$ as Decision Variables

Name	*Number of Rows/Columns*	*Number of Elements*	*Maximum Number of Nonzero Elements*
x	$N(K+1)-K$ / 1	-	-
H	$N(K+1)-K$ / $N(K+1)-K$	$N^2(K^2+2K+1)$ $+N(-2K^2-2K)+K^2$	$4NK-4K$ or, if $\mathbf{c}^+$ is simple, $4NJ-4J$
f	$N(K+1)-K$ / 1	$N(K+1)$ $-K$	1
A	$N(2K+I)-K$ / $N(K+1)-K$	$N^2(2K^2+IK+2K+I)$ $+N(-3K^2-IK-K)$ $+K^2$	$N(K^2+5K+I)$ $-K^2-4K$ or, if $\mathbf{P}$ is simple, $N(7K+I-J)$ $-6K+J$
b	$N(2K+I)-K$ / 1	$N(2K+I)$ $-K$	$K+I$
Total	-	$N^2(3K^2+IK+4K+I+1)$ $+N(-5K^2-IK+I+1)$ $+2K^2-2K$	$N(K^2+9K+I)$ $-K^2-7K+I+1$ or, if $\mathbf{P}$ is simple, $N(11K+I-J)$ $-9K+I+J+1$ or, if $\mathbf{c}^+$ is too, $N(7K+I+3J)$ $-5K+I-3J+1$

6.4 QUADRATIC PROGRAM (QP) FOR FLUID RELAXATION WITH $\Delta\mathbf{t}$ AND $\Delta\tilde{\mathbf{Q}}^+$ AS DECISION VARIABLES

If we substitute in $\Delta\mathbf{q}^+(t_n) \equiv \mathbf{q}^+(t_n) - \mathbf{q}^+(t_{n-1})$ which is the increase in the total (immediate and upstream) WIP in each time interval, we have Formulation 6.4, a QP with $\Delta\mathbf{t}$ and $\Delta\tilde{\mathbf{Q}}^+ \equiv \left[\Delta\tilde{\mathbf{q}}^+(t_1) \quad \Delta\tilde{\mathbf{q}}^+(t_2) \quad \cdots \quad \Delta\tilde{\mathbf{q}}^+(t_N)\right]$ as the primary decision variables and with $\Delta\tilde{\mathbf{U}}$, $\Delta\tilde{\mathbf{Y}}$, and $\tilde{\mathbf{Q}}$ as slack variables for their respective constraints (so they appear as objective function arguments but not under the "min").

Formulation 6.4: Quadratic Program (QP) for Fluid Relaxation with $\Delta\mathbf{t}$ and $\Delta\tilde{\mathbf{Q}}^+$ as Decision Variables

$$\min_{\Delta\mathbf{t},\Delta\tilde{\mathbf{Q}}^+} \tilde{C}_{\mathrm{h}}\left(\Delta\mathbf{t},\Delta\tilde{\mathbf{Q}}^+,\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}},\tilde{\mathbf{Q}} \mid N,...\right) = \left(\mathbf{c}^+\right)^{\mathrm{T}} \sum_{n=1}^{N} \Delta t_n \left[\mathbf{q}^+(t_0) - \frac{1}{2}\Delta\tilde{\mathbf{q}}^+(t_n) + \sum_{n'=1}^{n} \Delta\tilde{\mathbf{q}}^+(t_{n'})\right]$$

s.t.

$$-\boldsymbol{\alpha}^+ \Delta t_n + \Delta\tilde{\mathbf{q}}^+(t_n) + \Delta\tilde{\mathbf{u}}(t_n) = \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,...,N\}$$

$$-\boldsymbol{\chi}\Delta t_n - \mathbf{BD}(\mathbf{p})\Delta\tilde{\mathbf{q}}^+(t_n) + \Delta\tilde{\mathbf{y}}(t_n) = \mathbf{0}_{I,1} \qquad \forall n \in \{1,2,...,N\}$$

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\sum_{n'=1}^{n} \Delta\tilde{\mathbf{q}}^+(t_{n'}) + \tilde{\mathbf{q}}(t_n) = \mathbf{q}(t_0) \qquad \forall n \in \{1,2,...,N\}$$

$$\Delta t_n \geq 0 \qquad \forall n \in \{1,2,...,N\}$$

$$\Delta\tilde{\mathbf{q}}^+(t_n) \in \mathbb{R}^K \left(\text{unrestricted}\right) \qquad \forall n \in \{1,2,...,N\}$$

$$\tilde{\mathbf{q}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,...,N-1\}$$

$$\tilde{\mathbf{q}}(t_N) = \mathbf{0}_{K,1}$$

$$\Delta\tilde{\mathbf{u}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,...,N\}$$

$$\Delta\tilde{\mathbf{y}}(t_n) \geq \mathbf{0}_{I,1} \qquad \forall n \in \{1,2,...,N\}$$

This QP formulation can also be represented more compactly as shown in Formulation 6.5:

Formulation 6.5: Quadratic Program (QP) for Fluid Relaxation with $\Delta\mathbf{t}$ and $\Delta\tilde{\mathbf{Q}}^{+}$ as Decision Variables (in Compact Form)

$$\min_{\Delta\mathbf{t},\Delta\tilde{\mathbf{Q}}^{+}} \tilde{C}_{\mathrm{h}}\left(\Delta\mathbf{t},\Delta\tilde{\mathbf{Q}}^{+},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}},\tilde{\mathbf{Q}} \mid N,\ldots\right) = \left(\mathbf{c}^{+}\right)^{\mathrm{T}}\left[\mathbf{q}^{+}(t_0)\mathbf{1}_N^{\mathrm{T}} + \Delta\tilde{\mathbf{Q}}^{+}\left(\mathbf{L}^{\mathrm{T}} - \frac{1}{2}\mathbf{I}_N\right)\right]\Delta\mathbf{t}$$

s.t.
$$-\boldsymbol{\alpha}^{+}\Delta\mathbf{t}^{\mathrm{T}} + \Delta\tilde{\mathbf{Q}}^{+} + \Delta\tilde{\mathbf{U}} = \mathbf{0}_{K,N}$$
$$-\boldsymbol{\chi}\Delta\mathbf{t}^{\mathrm{T}} - \mathbf{B}\mathbf{D}(\mathbf{p})\Delta\tilde{\mathbf{Q}}^{+} + \Delta\tilde{\mathbf{Y}} = \mathbf{0}_{I,N}$$
$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\Delta\tilde{\mathbf{Q}}^{+}\mathbf{L} + \tilde{\mathbf{Q}} = \mathbf{q}(t_0)\mathbf{1}_N^{\mathrm{T}}$$
$$\tilde{\mathbf{Q}}\mathbf{e}_N = \mathbf{0}_{K,1}$$
$$\Delta\mathbf{t} \geq \mathbf{0}_{N,1}$$
$$\tilde{\mathbf{Q}} \geq \mathbf{0}_{K,N}$$
$$\Delta\tilde{\mathbf{U}} \geq \mathbf{0}_{K,N}$$
$$\Delta\tilde{\mathbf{Y}} \geq \mathbf{0}_{I,N}$$

Here $\mathbf{1}_N^{\mathrm{T}}$ is a transposed vector of all ones, $\mathbf{e}_N$ is a unit vector of all zeros except for a one in the *N*th position, and $\mathbf{L}$ is a unit lower triangular matrix of all ones. Matrix inequalities (such as $\tilde{\mathbf{Q}} \geq \mathbf{0}_{K,N}$) are meant to imply that each element of the matrix is nonnegative and not that the matrix as a whole is positive semi-definite.

The vectors and matrices in equations (6.1), (6.3), and (6.4) now have the following form:

$$\mathbf{x} = \begin{bmatrix} \Delta\mathbf{t} \\ \mathbf{x}_{\Delta\tilde{\mathbf{Q}}^{+}} \end{bmatrix},\ \overset{\geq}{\mathbf{x}} = \Delta\mathbf{t},\ \mathbf{f} = \begin{bmatrix} \mathbf{f}_{\Delta\mathbf{t}} \\ \mathbf{0}_{NK,1} \end{bmatrix},\ \mathbf{H} = \begin{bmatrix} \mathbf{0}_{N,N} & \mathbf{H}_{\Delta\tilde{\mathbf{Q}}^{+},\Delta\mathbf{t}}^{\mathrm{T}} \\ \mathbf{H}_{\Delta\tilde{\mathbf{Q}}^{+},\Delta\mathbf{t}} & \mathbf{0}_{NK,NK} \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{\Delta\mathbf{U},\Delta\mathbf{t}} & \mathbf{A}_{\Delta\mathbf{U},\Delta\tilde{\mathbf{Q}}^{+}} \\ \mathbf{A}_{\Delta\mathbf{Y},\Delta\mathbf{t}} & \mathbf{A}_{\Delta\mathbf{Y},\Delta\tilde{\mathbf{Q}}^{+}} \\ \mathbf{0}_{(N-1)K,N} & \mathbf{A}_{\mathbf{Q},\Delta\tilde{\mathbf{Q}}^{+}} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{b}_{\Delta\mathbf{U}} \\ \mathbf{b}_{\Delta\mathbf{Y}} \\ \mathbf{b}_{\mathbf{Q}} \end{bmatrix},$$

where the submatrices are as follows (with blank subsubmatrices indicating zeros and with lines separating subsubmatrices corresponding to different values of *n*):

$$\mathbf{x}_{\Delta\tilde{\mathbf{Q}}^+} = \begin{bmatrix} \Delta\tilde{\mathbf{q}}^+(t_1) \\ \hline \Delta\tilde{\mathbf{q}}^+(t_2) \\ \hline \vdots \\ \hline \Delta\tilde{\mathbf{q}}^+(t_N) \end{bmatrix}, \quad \mathbf{f}_{\Delta\mathbf{t}} = \mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)\mathbf{1}_N,$$

$$\mathbf{H}_{\Delta\tilde{\mathbf{Q}}^+,\Delta\mathbf{t}} = \left[\begin{array}{c|c|c|c|c|c} \mathbf{c}^+/2 & \mathbf{c}^+ & \mathbf{c}^+ & \cdots & \mathbf{c}^+ & \mathbf{c}^+ \\ \hline & \mathbf{c}^+/2 & \mathbf{c}^+ & \cdots & \mathbf{c}^+ & \mathbf{c}^+ \\ \hline & & \mathbf{c}^+/2 & \ddots & \vdots & \vdots \\ \hline & & & \ddots & \mathbf{c}^+ & \mathbf{c}^+ \\ \hline & & & & \mathbf{c}^+/2 & \mathbf{c}^+ \\ \hline & & & & & \mathbf{c}^+/2 \end{array}\right],$$

$$\mathbf{A}_{\Delta\mathbf{U},\Delta\mathbf{t}} = \begin{bmatrix} -\boldsymbol{\alpha}^+ & & & \\ \hline & -\boldsymbol{\alpha}^+ & & \\ \hline & & \ddots & \\ \hline & & & -\boldsymbol{\alpha}^+ \end{bmatrix}, \quad \mathbf{A}_{\Delta\mathbf{U},\Delta\tilde{\mathbf{Q}}^+} = \begin{bmatrix} \mathbf{I}_K & & & \\ \hline & \mathbf{I}_K & & \\ \hline & & \ddots & \\ \hline & & & \mathbf{I}_K \end{bmatrix},$$

$$\mathbf{A}_{\Delta\mathbf{Y},\Delta\mathbf{t}} = \begin{bmatrix} -\boldsymbol{\chi} & & & \\ \hline & -\boldsymbol{\chi} & & \\ \hline & & \ddots & \\ \hline & & & -\boldsymbol{\chi} \end{bmatrix}, \quad \mathbf{A}_{\Delta\mathbf{Y},\Delta\tilde{\mathbf{Q}}^+} = \begin{bmatrix} -\mathbf{BD}(\mathbf{p}) & & & \\ \hline & -\mathbf{BD}(\mathbf{p}) & & \\ \hline & & \ddots & \\ \hline & & & -\mathbf{BD}(\mathbf{p}) \end{bmatrix},$$

$$\mathbf{A}_{\mathbf{Q},\Delta\tilde{\mathbf{Q}}^+} = \begin{bmatrix} -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) & & & \\ \hline -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) & & \\ \hline \vdots & \vdots & \ddots & \\ \hline -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) & \cdots & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) \end{bmatrix}, \text{ and } \mathbf{b}_{\mathbf{Q}} = \begin{bmatrix} \mathbf{q}(t_0) \\ \hline \mathbf{q}(t_0) \\ \hline \vdots \\ \hline \mathbf{q}(t_0) \end{bmatrix}.$$

These vectors and matrices have the dimensions and sparsity shown in Table 6.4:

Table 6.4: Dimensions and Sparsity of Vectors and Matrices in QP Formulation with Δt and $\Delta\tilde{\mathbf{Q}}^+$ as Decision Variables

Name	*Number of Rows/Columns*	*Number of Elements*	*Maximum Number of Nonzero Elements*
x	$N(K+1)$ / 1	-	-
H	$N(K+1)$ / $N(K+1)$	$N^2(K^2+2K+1)$	$N^2K/2+NK/2$ or, if $\mathbf{c}^+$ is simple, $N^2J/2+NJ/2$
f	$N(K+1)$ / 1	$N(K+1)$	N
A	$N(2K+I)$ / $N(K+1)$	$N^2(2K^2+IK+2K+I)$	$N^2K^2/2$ $+N(K^2/2+3K+I)$ or, if **P** is simple, $N^2(K-J/2)$ $+N(4K+I-J/2)$
b	$N(2K+I)$ / 1	$N(2K+I)$	NK
Total	-	$N^2(3K^2+IK+4K+I+1)$ $+N(4K+I+2)$	$N^2(K^2+K)/2$ $+N(K^2/2+9K/2+I+1)$ or, if **P** is simple, $N^2(3K-J)/2$ $+N(11K/2+I-J/2+1)$ or, if $\mathbf{c}^+$ is too, $N^2K+N(5K+I+1)$

In the author's limited experience, the minimal number of breakpoints N required to give an optimal solution to the SCLP formulation is close to the number of queues K. If we compare the different QP formulations under this assumption, we can see that the formulation with $\Delta\mathbf{t}$ and $\tilde{\mathbf{Q}}^+$ as the primary decision variables is superior in terms of matrix size and sparsity, so that will be the QP formulation on which we build from here on.

6.5 DUAL QUADRATIC PROGRAM (DQP) FOR FLUID RELAXATION

A quadratic program of the form given by equations (6.1), (6.3), and (6.4) has a dual quadratic program of the form

$$\min_{\mathbf{x},\mathbf{z}} \left(\frac{\mathbf{x}^{\mathrm{T}}\mathbf{H}\mathbf{x}}{2} + \mathbf{b}^{\mathrm{T}}\mathbf{z} \right)$$

$$\text{s.t.} \quad -\mathbf{H}\mathbf{x} - \mathbf{A}^{\mathrm{T}}\mathbf{z} \le \mathbf{f} \tag{6.15}$$

$$\overset{\geq}{\mathbf{x}} \ge \mathbf{0}$$

$$\mathbf{z} \ge \mathbf{0}$$

where $\overset{\geq}{\mathbf{x}}$ is the subvector of $\mathbf{x}$ that is constrained to be nonnegative and constraint (6.15) is actually an equality for those rows corresponding to unrestricted decision variables in $\mathbf{x}$.

Filling in the vectors and matrices gives Formulation 6.6, a dual quadratic program (DQP) where the primary decision variables are $\Delta\mathbf{t}$, $\underline{\tilde{\mathbf{Q}}}$, $\boldsymbol{\Xi} \equiv \left[\boldsymbol{\xi}(t_1) \quad \boldsymbol{\xi}(t_2) \quad \cdots \quad \boldsymbol{\xi}(t_N)\right]$, $\boldsymbol{\Omega} \equiv \left[\boldsymbol{\omega}(t_1) \quad \boldsymbol{\omega}(t_2) \quad \cdots \quad \boldsymbol{\omega}(t_N)\right]$, and (although it does not appear in the objective function) $\boldsymbol{\Gamma} \equiv \left[\boldsymbol{\gamma}(t_1) \quad \boldsymbol{\gamma}(t_2) \quad \cdots \quad \boldsymbol{\gamma}(t_{N-1})\right]$ while the slack variables for the first set of constraints are $\boldsymbol{\delta} \equiv \left[\delta_1 \quad \delta_2 \quad \cdots \quad \delta_N\right]^{\mathrm{T}}$.

Formulation 6.6: Dual Quadratic Program (DQP) for Fluid Relaxation

$$\min_{\Delta\mathbf{t},\tilde{\mathbf{Q}}^{+},\boldsymbol{\Omega},\boldsymbol{\Xi},\boldsymbol{\Gamma}} \tilde{C}'_{\mathrm{h}}\left(\Delta\mathbf{t},\underline{\tilde{\mathbf{Q}}},\boldsymbol{\Omega},\boldsymbol{\Xi},\boldsymbol{\Gamma},\boldsymbol{\delta}\,|\,N,...\right)$$

$$= \mathbf{q}^{+}(t_0)^{\mathrm{T}}\left[\boldsymbol{\omega}(t_1) - \mathbf{D}(\mathbf{p})\mathbf{B}^{\mathrm{T}}\boldsymbol{\xi}(t_n)\right] + \frac{\left(\mathbf{c}^{+}\right)^{\mathrm{T}}}{2}\sum_{n=1}^{N-1}\tilde{\mathbf{q}}^{+}(t_n)\left(\Delta t_n + \Delta t_{n+1}\right)$$

s.t.

$$+\left(\boldsymbol{\alpha}^{+}\right)^{\mathrm{T}}\boldsymbol{\omega}(t_1) + \boldsymbol{\chi}^{\mathrm{T}}\boldsymbol{\xi}(t_1) - \frac{\left(\mathbf{c}^{+}\right)^{\mathrm{T}}}{2}\tilde{\mathbf{q}}^{+}(t_1) + \delta_1 = \frac{\mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)}{2}$$

$$+\left(\boldsymbol{\alpha}^{+}\right)^{\mathrm{T}}\boldsymbol{\omega}(t_n) + \boldsymbol{\chi}^{\mathrm{T}}\boldsymbol{\xi}(t_n) - \frac{\left(\mathbf{c}^{+}\right)^{\mathrm{T}}}{2}\left[\tilde{\mathbf{q}}^{+}(t_n) + \tilde{\mathbf{q}}^{+}(t_{n-1})\right] + \delta_n = 0 \quad \forall n \in \{2,3,...,N-1\}$$

$$+\left(\boldsymbol{\alpha}^{+}\right)^{\mathrm{T}}\boldsymbol{\omega}(t_N) + \boldsymbol{\chi}^{\mathrm{T}}\boldsymbol{\xi}(t_N) - \frac{\left(\mathbf{c}^{+}\right)^{\mathrm{T}}}{2}\tilde{\mathbf{q}}^{+}(t_{N-1}) + \delta_N = 0$$

$$-\boldsymbol{\omega}(t_n) + \boldsymbol{\omega}(t_{n+1}) + \mathbf{D}(\mathbf{p})\mathbf{B}^{\mathrm{T}}\left[\boldsymbol{\xi}(t_n) - \boldsymbol{\xi}(t_{n+1})\right]$$

$$-\frac{\mathbf{c}^{+}}{2}\left(\Delta t_n + \Delta t_{n+1}\right) + \left(\mathbf{I}_K - \mathbf{P}\right)\boldsymbol{\gamma}(t_n) = \mathbf{0}_{K,1} \quad \forall n \in \{1,2,...,N-1\}$$

$$\Delta t_n \geq 0 \quad \forall n \in \{1,2,...,N\}$$

$$\tilde{\mathbf{q}}^{+}(t_n) \in \mathbb{R}^K \text{ (unrestricted)} \quad \forall n \in \{1,2,...,N-1\}$$

$$\boldsymbol{\omega}(t_n) \geq \mathbf{0}_{K,1} \quad \forall n \in \{1,2,...,N\}$$

$$\boldsymbol{\xi}(t_n) \geq \mathbf{0}_{I,1} \quad \forall n \in \{1,2,...,N\}$$

$$\boldsymbol{\gamma}(t_n) \geq \mathbf{0}_{K,1} \quad \forall n \in \{1,2,...,N-1\}$$

$$\delta_n \geq 0 \quad \forall n \in \{1,2,...,N\}$$

If **H** were positive semi-definite, then strong duality would hold and the Complementary Slackness Principle would apply. That principle says that if $\Delta\mathbf{t}^*$, $\tilde{\mathbf{Q}}^{+*}$, $\Delta\tilde{\mathbf{U}}^*$, $\Delta\tilde{\mathbf{Y}}^*$, and $\tilde{\mathbf{Q}}^*$ form a feasible solution to the primal quadratic program (QP), and if $\Delta\mathbf{t}^*$, $\tilde{\mathbf{Q}}^{+*}$, $\boldsymbol{\Omega}^*$, $\boldsymbol{\Xi}^*$, $\boldsymbol{\Gamma}^*$, and $\boldsymbol{\delta}^*$ form a feasible solution to the dual quadratic program (DQP), and if

$$\sum_{n=1}^{N-1}\boldsymbol{\gamma}^*(t_n)^{\mathrm{T}}\tilde{\mathbf{q}}^*(t_n) + \sum_{n=1}^{N}\left[\delta_n^*\Delta t_n^* + \boldsymbol{\omega}^*(t_n)^{\mathrm{T}}\Delta\tilde{\mathbf{u}}^*(t_n) + \boldsymbol{\xi}^*(t_n)^{\mathrm{T}}\Delta\tilde{\mathbf{y}}^*(t_n)\right] = 0\,, \tag{6.16}$$

then $\Delta\mathbf{t}^*$, $\tilde{\mathbf{Q}}^{+*}$, $\Delta\tilde{\mathbf{U}}^*$, $\Delta\tilde{\mathbf{Y}}^*$, and $\tilde{\mathbf{Q}}^*$ form an optimal solution to the primal quadratic program (QP), while $\Delta\mathbf{t}^*$, $\tilde{\mathbf{Q}}^{+*}$, $\boldsymbol{\Omega}^*$, $\boldsymbol{\Xi}^*$, $\boldsymbol{\Gamma}^*$, and $\boldsymbol{\delta}^*$ form an optimal solution to the dual quadratic program (DQP). Note that since all of the variables in equation (6.16) must be nonnegative, this is equivalent to element-by-element complementary slackness:

$$\delta_n^* \Delta t_n^* = 0 \qquad \forall n \in \{1,2,...,N\},$$

$$\gamma_{j,k}^*(t_n)\tilde{q}_{j,k}^*(t_n) = 0 \qquad \begin{cases} \forall n \in \{1,2,...,N-1\} \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \end{cases},$$

$$\omega_{j,k}^*(t_n)\Delta\tilde{u}_{j,k}^*(t_n) = 0 \qquad \begin{cases} \forall n \in \{1,2,...,N\} \\ \forall j \in \{1,2,...,J\} \\ \forall k \in \{1,2,...,K_j\} \end{cases},$$

and

$$\xi_i^*(t_n)\Delta\tilde{y}_i^*(t_n) = 0 \qquad \begin{cases} \forall n \in \{1,2,...,N\} \\ \forall i \in \{1,2,...,I\} \end{cases}.$$

6.6 NONCONVEXITY OF QUADRATIC PROGRAM (QP) FORMULATIONS

Unfortunately, $\mathbf{H}$ is not positive semi-definite in any of these QP formulations. Thus, the objective function is not convex, and we are not guaranteed strong duality (so there may be a duality gap). Without convexity, global minima are not guaranteed to be found by standard QP solvers. Local minima can be found either by an interior-point method or by an active-set method that finds a solution to the following linear set of equations:

$$\begin{bmatrix} \mathbf{H} & \bar{\bar{\mathbf{A}}}^{T} & \bar{\mathbf{A}}^{T} \\ \bar{\bar{\mathbf{A}}} & \mathbf{0} & \mathbf{0} \\ \bar{\mathbf{A}} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \bar{\bar{\boldsymbol{\lambda}}} \\ \bar{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} -\mathbf{f} \\ \bar{\bar{\mathbf{b}}} \\ \bar{\mathbf{b}} \end{bmatrix}$$

where the $\boldsymbol{\lambda}$ vectors contain the Lagrange multipliers corresponding to the active (binding) constraints with $\bar{\mathbf{A}}$ and $\bar{\mathbf{b}}$ containing the rows of **A** and **b** that are active (at their bounds).

However, the Branch-And-Reduce Optimization Navigator (BARON) developed by Sahinidis [1996] and others is a new GAMS solver for the global solution of nonlinear (NLP) and mixed-integer nonlinear programs (MINLP). While traditional NLP and MINLP algorithms are guaranteed to converge only under certain convexity assumptions, BARON implements deterministic global optimization algorithms of the branch-and-bound type that are guaranteed to provide global optima under fairly general assumptions. These include the availability of finite lower and upper bounds on the variables and their expressions in the NLP or MINLP to be solved. BARON implements algorithms of the branch-and-bound type enhanced with a variety of constraint propagation and duality techniques for reducing ranges of variables in the course of the algorithm.

6.7 EXAMPLE NETWORK: QP SOLUTIONS

The sequence of solutions from the QP formulations are plotted in the following figures for the example network. For each number of time intervals *N*, the initial feasible solution used as a starting point for the QP formulation is an

interpolation of the optimal makespan solution with the interpolating breakpoints given by:

$$t_n = \left(\frac{n}{N}\right)^{2.4} T^*.$$

The exponent was chosen to approximate where the interpolating breakpoints tended to cluster in the optimal QP solutions (in the author's limited experience).

Figures 6.1 and 6.2 show the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ and the machine capacity allocation $\tilde{v}^{*}_{j,k}(t)$ over time for the initial feasible solution when $N = 2$. Figures 6.3 and 6.4 show the same for the optimal solution when $N = 2$ where the improvement in the objective function value is

$$\frac{\tilde{C}^*_{\mathrm{h}}(\Delta\mathbf{t}, \tilde{\mathbf{Q}}, \Delta\tilde{\mathbf{U}} \mid N = 2, \ldots)}{\tilde{C}^*_{\mathrm{h}}(\Delta\mathbf{t}, \tilde{\mathbf{Q}}, \Delta\tilde{\mathbf{U}} \mid N = 1, \ldots)} = \frac{932.7}{1152} \approx 0.8096354167\,.$$

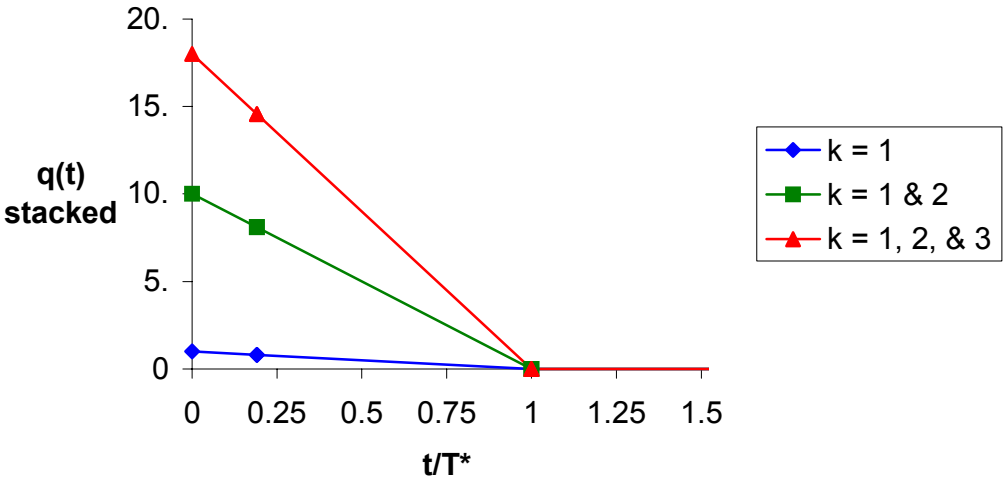

Figure 6.1: Example Total WIP in $N = 2$ QP Feasible Solution

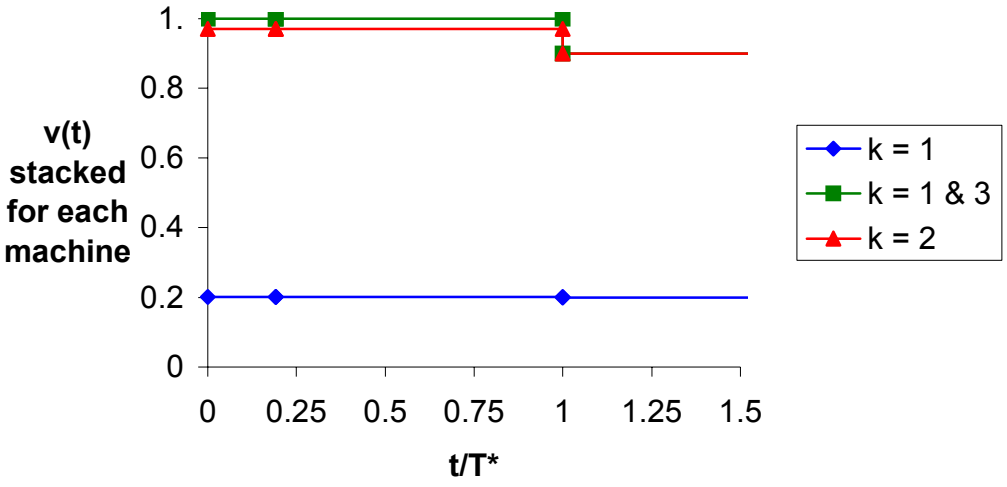

Figure 6.2: Example Machine Utilization in $N = 2$ QP Feasible Solution

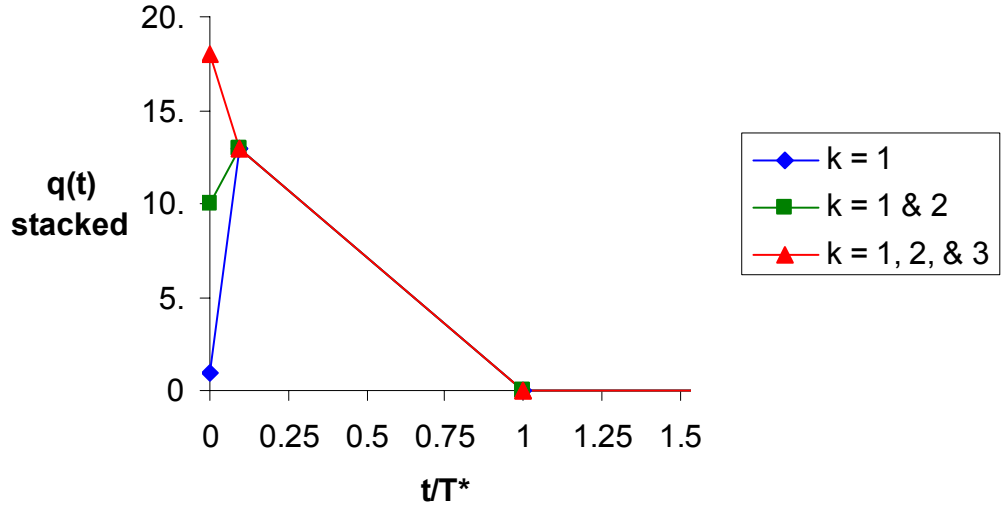

Figure 6.3: Example Total WIP in $N = 2$ QP Optimal Solution

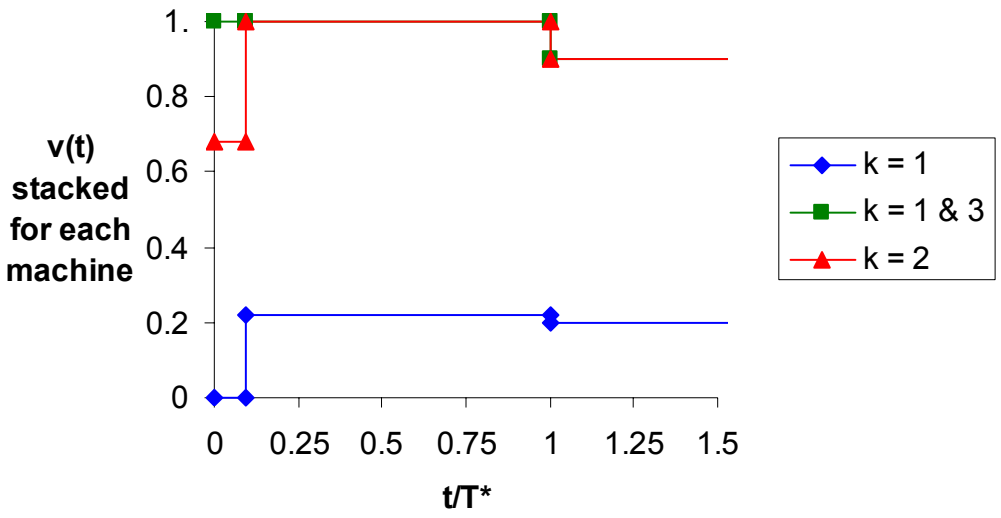

Figure 6.4: Example Machine Utilization in $N = 2$ QP Optimal Solution

Figures 6.5 and 6.6 show the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ and the machine capacity allocation $\tilde{v}^{*}_{j,k}(t)$ over time for the initial feasible solution when $N = 3$. Figures 6.7 and 6.8 show the same for the optimal solution when $N = 3$. This objective function value is the same as for $N = 2$ and the same as the SCLP optimal solution given by Weiss' algorithm; the only difference between the three optimal solutions is how fast the second buffer drains. The $N = 3$ optimal solution is also the optimal solution when N is larger than three; either some time intervals have zero width or adjacent time intervals have the same processing rates.

It should also be noted that all of these optimal solutions have the system drain at time T^* even though the QP formulations allow the drain time to vary. Of 30,000 instances of this example network with randomly generated $\mathbf{p}$, $\mathbf{c}$, $\boldsymbol{\alpha}$, and $\mathbf{q}(t_0)$, only 7% of the optimal solutions had the system drain after time T^*.

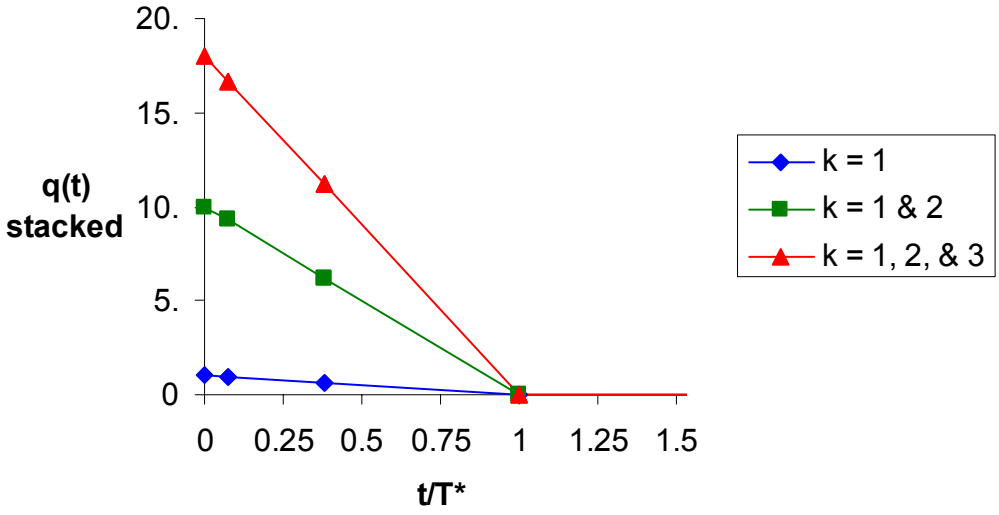

Figure 6.5: Example Total WIP in $N = 3$ QP Feasible Solution

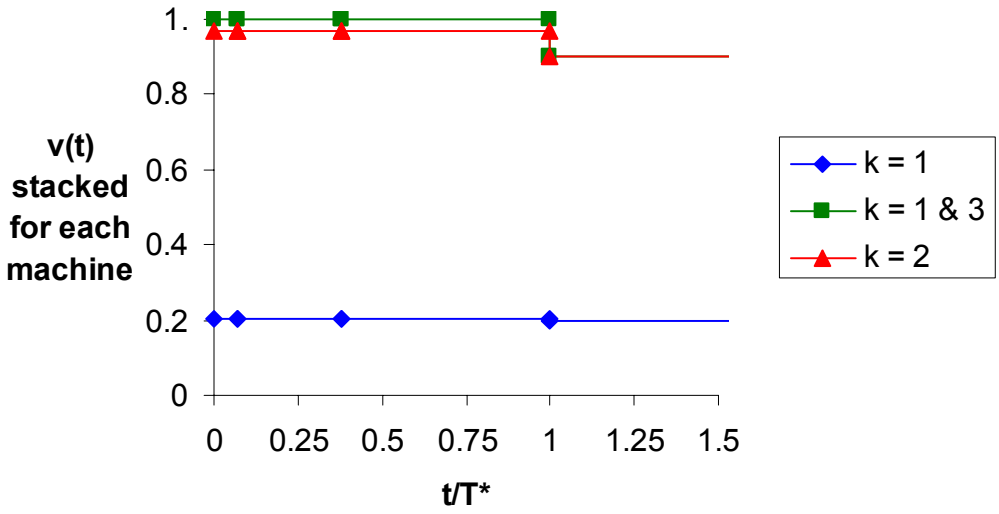

Figure 6.6: Example Machine Utilization in $N = 3$ QP Feasible Solution

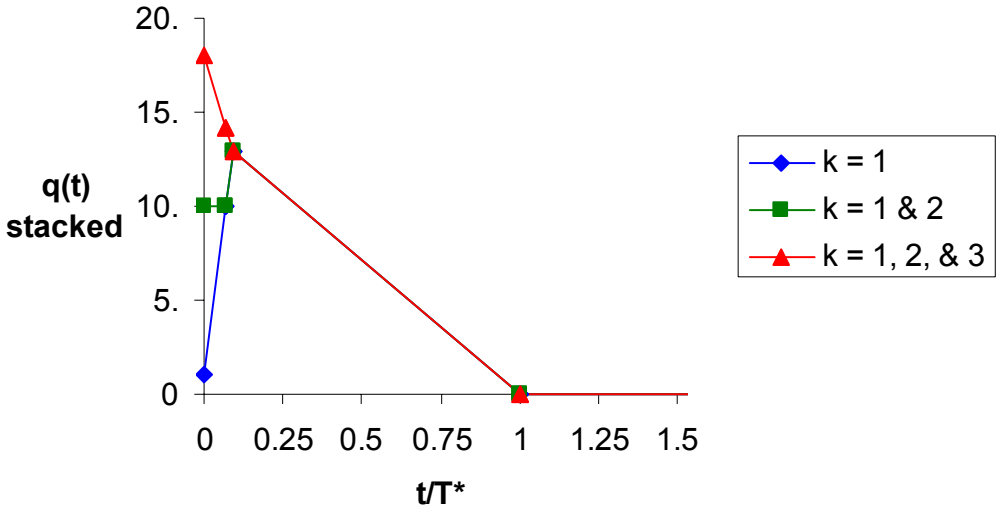

Figure 6.7: Example Total WIP in $N = 3$ QP Optimal Solution

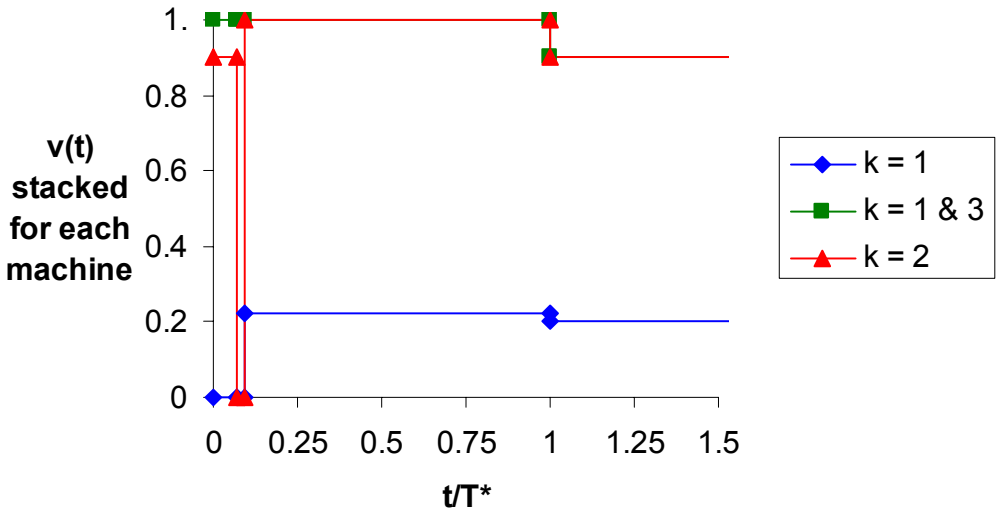

Figure 6.8: Example Machine Utilization in $N = 3$ QP Optimal Solution

7. Linear Programming (LP) Models: Fluid Relaxation

The QP formulation is, in fact, a bilinear program. This means that the decision variables can be partitioned into two sets such that if the variables in either set are fixed at a constant value, the remaining constraints and the objective function are linear in the decision variables. Such a formulation, with one set of variables having been fixed, is a linear program of the form

$$\min_{\mathbf{x}} \left(\mathbf{f}^{\mathrm{T}} \mathbf{x} + c \right) \tag{7.1}$$

$$\text{s.t.} \quad \overset{=}{\mathbf{A}}\,\mathbf{x} = \overset{=}{\mathbf{b}} \tag{7.2}$$

$$\mathbf{A}\mathbf{x} \le \mathbf{b} \tag{7.3}$$

$$\overset{\geq}{\mathbf{x}} \ge \mathbf{0}\,. \tag{7.4}$$

7.1 Linear Program (LP) for Fluid Relaxation with Fixed Time Intervals

One obvious way to partition the decision variables is between $\Delta\mathbf{t}$ and $\tilde{\mathbf{Q}}^{+}$. If $\Delta\mathbf{t}$ is held fixed (but feasible), then the vectors and matrices in equations (7.1) through (7.4) have the following form:

$$\mathbf{x} = \mathbf{x}_{\tilde{\mathbf{Q}}^{+}},\ \overset{\geq}{\mathbf{x}} = \varnothing,\ \mathbf{f} = \mathbf{H}_{\tilde{\mathbf{Q}}^{+},\Delta\mathbf{t}}\Delta\mathbf{t},\ c = \mathbf{f}_{\Delta\mathbf{t}}^{\mathrm{T}}\Delta\mathbf{t},\ \overset{=}{\mathbf{A}} = \varnothing,\ \overset{=}{\mathbf{b}} = \varnothing,$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{\Delta\mathbf{U},\tilde{\mathbf{Q}}^{+}} \\ \mathbf{A}_{\Delta\mathbf{Y},\tilde{\mathbf{Q}}^{+}} \\ \mathbf{A}_{\mathbf{Q},\tilde{\mathbf{Q}}^{+}} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{b}_{\Delta\mathbf{U}} \\ \mathbf{b}_{\Delta\mathbf{Y}} \\ \mathbf{0}_{(N-1)K,1} \end{bmatrix} - \begin{bmatrix} \mathbf{A}_{\Delta\mathbf{U},\Delta\mathbf{t}} \\ \mathbf{A}_{\Delta\mathbf{Y},\Delta\mathbf{t}} \\ \mathbf{0}_{(N-1)K,N} \end{bmatrix} \Delta\mathbf{t}\,,$$

where the submatrices are as they were defined in equations (6.9) through (6.14). Filling in the vectors and matrices gives Formulation 7.1:

Formulation 7.1: Linear Program (LP) for Fluid Relaxation with Fixed Time Intervals

$$\min_{\tilde{\mathbf{Q}}^+} \tilde{C}_{\mathrm{h}}\left(\tilde{\mathbf{Q}}^+, \Delta\tilde{\mathbf{U}}, \Delta\tilde{\mathbf{Y}}, \tilde{\mathbf{Q}} \mid N, \Delta\mathbf{t}, \ldots\right) = \frac{\left(\mathbf{c}^+\right)^{\mathrm{T}}}{2}\left[\mathbf{q}^+(t_0)\Delta t_1 + \sum_{n=1}^{N-1} \tilde{\mathbf{q}}^+(t_n)\left(\Delta t_n + \Delta t_{n+1}\right)\right]$$

s.t.

$$\tilde{\mathbf{q}}^+(t_1) + \Delta\tilde{\mathbf{u}}(t_1) = \mathbf{q}^+(t_0) + \boldsymbol{\alpha}^+\Delta t_1$$

$$-\tilde{\mathbf{q}}^+(t_{n-1}) + \tilde{\mathbf{q}}^+(t_n) + \Delta\tilde{\mathbf{u}}(t_n) = \boldsymbol{\alpha}^+\Delta t_n \qquad \forall n \in \{2,3,\ldots,N-1\}$$

$$-\tilde{\mathbf{q}}^+(t_{N-1}) + \Delta\tilde{\mathbf{u}}(t_N) = \boldsymbol{\alpha}^+\Delta t_N$$

$$-\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^+(t_1) + \Delta\tilde{\mathbf{y}}(t_1) = -\mathbf{BD}(\mathbf{p})\mathbf{q}^+(t_0) + \boldsymbol{\chi}\Delta t_1$$

$$\mathbf{BD}(\mathbf{p})\left[\tilde{\mathbf{q}}^+(t_{n-1}) - \tilde{\mathbf{q}}^+(t_n)\right] + \Delta\tilde{\mathbf{y}}(t_n) = \boldsymbol{\chi}\Delta t_n \qquad \forall n \in \{2,3,\ldots,N-1\}$$

$$\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^+(t_{N-1}) + \Delta\tilde{\mathbf{y}}(t_N) = \boldsymbol{\chi}\Delta t_N$$

$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\tilde{\mathbf{q}}^+(t_n) + \tilde{\mathbf{q}}(t_n) = \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,\ldots,N-1\}$$

$$\tilde{\mathbf{q}}^+(t_n) \in \mathbb{R}^K \left(\text{unrestricted}\right) \qquad \forall n \in \{1,2,\ldots,N-1\}$$

$$\tilde{\mathbf{q}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,\ldots,N-1\}$$

$$\Delta\tilde{\mathbf{u}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,\ldots,N\}$$

$$\Delta\tilde{\mathbf{y}}(t_n) \geq \mathbf{0}_{I,1} \qquad \forall n \in \{1,2,\ldots,N\}$$

These vectors and matrices have the dimensions and sparsity shown in Table 7.1:

Table 7.1: Dimensions and Sparsity of Vectors and Matrices in LP Formulation with Fixed Time Intervals

Name	*Number of Rows/Columns*	*Number of Elements*	*Maximum Number of Nonzero Elements*
x	$NK-K$ / 1	-	-
c	1/1	1	1
f	$NK-K$ / 1	$NK-K$	$NK-K$ or, if $\mathbf{c}^+$ is simple, $NJ-J$
A	$N(2K+I)-K$ / $NK-K$	$N^2(2K^2+IK)$ $+N(-3K^2-IK)$ $+K^2$	$N(K^2+4K)$ $-K^2-4K$ or, if **P** is simple, $N(6K-J)$ $-6K+J$
b	$N(2K+I)-K$ / 1	$N(2K+I)-K$	$N(K+I)$
Total	-	$N^2(2K^2+IK)$ $+N(-3K^2-IK+4K+I)$ $+K^2-3K$	$N(K^2+6K+I)$ $-K^2-5K+1$ or, if **P** is simple, $N(8K+I-J)$ $-7K+J+1$ or, if $\mathbf{c}^+$ is too, $N(7K+I)$ $-6K+1$

For sufficiently large N, a globally optimal solution to the QP formulation gives an exact optimal solution to the CLP formulation. Since this alternative LP formulation constitutes a numerical integration approach to solving the CLP, Pullan [2002], Fleischer and Sethuraman [2003], and others have shown that it converges asymptotically to the optimal solution to the CLP formulation as N increases and $\Delta\mathbf{t}$ decreases.

For $N=1$ and $\Delta t_1 = T^*$, the optimal solution to the weighted holding cost problem in this LP formulation is identical to the optimal makespan solution. For

each $N \geq 1$, the optimal solution to the weighted holding cost problem in the LP formulation can be transformed into a feasible solution for N twice as large and $\Delta\mathbf{t}$ cut in half. This suggests the following algorithm for finding the optimal solution to the weighted holding cost problem in the CLP formulation:

1. Begin with $N = 1$ and $\Delta t = T^*$ and the optimal makespan solution.
2. Double N and cut Δt exactly in half and solve the LP formulation with the transformed prior optimal solution as the new beginning feasible solution.
3. If the new optimal objective function value is not sufficiently better than the prior value, stop. Otherwise, adjust N so that the system does not drain until the beginning of the only the last time interval (so $\Delta\tilde{\mathbf{u}}(t_N) = \mathbf{0}_{K,1}$ and $\Delta\tilde{\mathbf{u}}(t_n) \neq \mathbf{0}_{K,1} \forall n < N$), and go to step 2.

The sequence of solutions from this LP formulation are plotted in the following figures for the example network. Figures 7.1 and 7.2 show the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ and the machine capacity allocation $\tilde{v}^*_{j,k}(t)$ over time for the initial feasible solution when $\Delta t = T^*/2$. Figures 7.3 and 7.4 show the same for the optimal solution when $\Delta t = T^*/2$ where the improvement in the objective function value is

$$\frac{\tilde{C}^*_{\mathrm{h}}(\tilde{\mathbf{Q}}^+, \Delta\tilde{\mathbf{U}}, \Delta\tilde{\mathbf{Y}}, \tilde{\mathbf{Q}} \mid \Delta t = T^*/2, \ldots)}{\tilde{C}^*_{\mathrm{h}}(\tilde{\mathbf{Q}}^+, \Delta\tilde{\mathbf{U}}, \Delta\tilde{\mathbf{Y}}, \tilde{\mathbf{Q}} \mid \Delta t = T^*, \ldots)} = \frac{1031.\overline{1}}{1152} \approx 0.8950617284 .$$

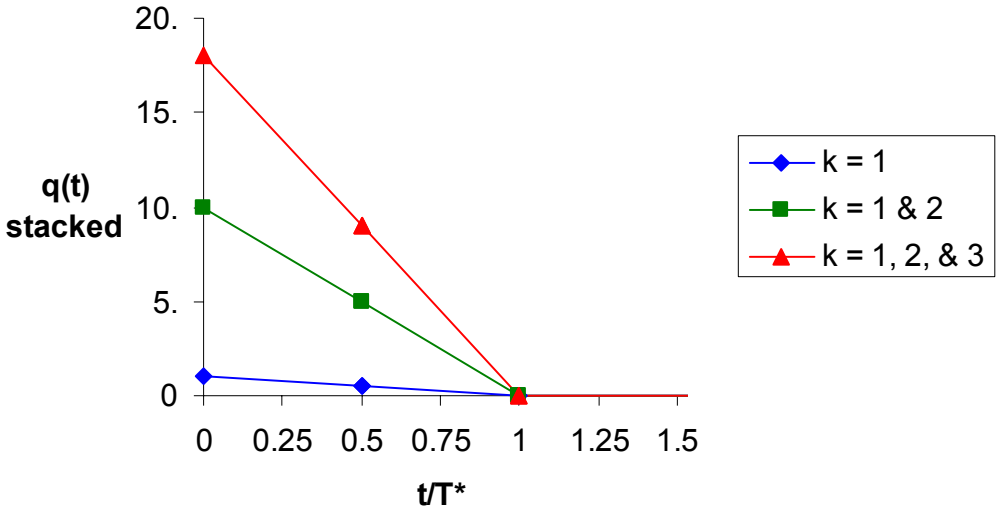

Figure 7.1: Example Total WIP in $\Delta t = T^*/2$ LP Feasible Solution

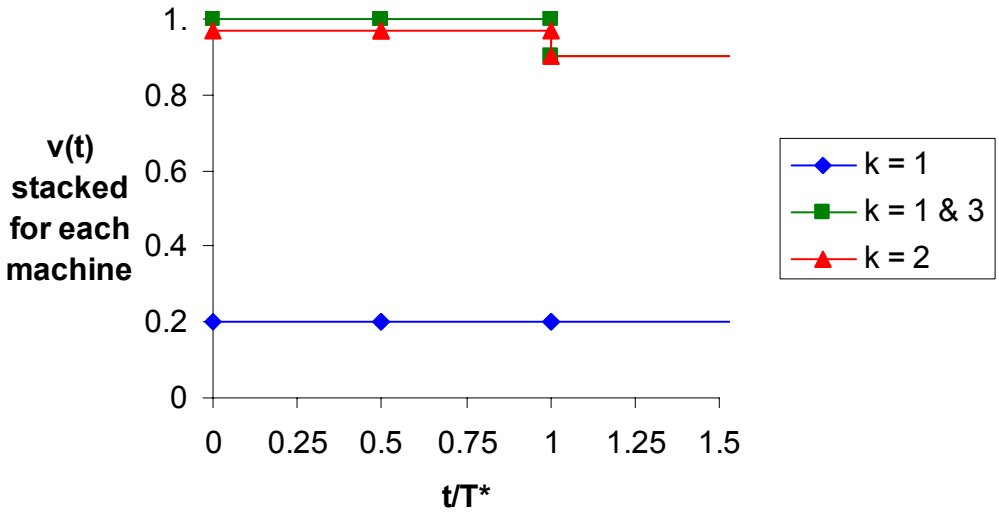

Figure 7.2: Example Machine Utilization in $\Delta t = T^*/2$ LP Feasible Solution

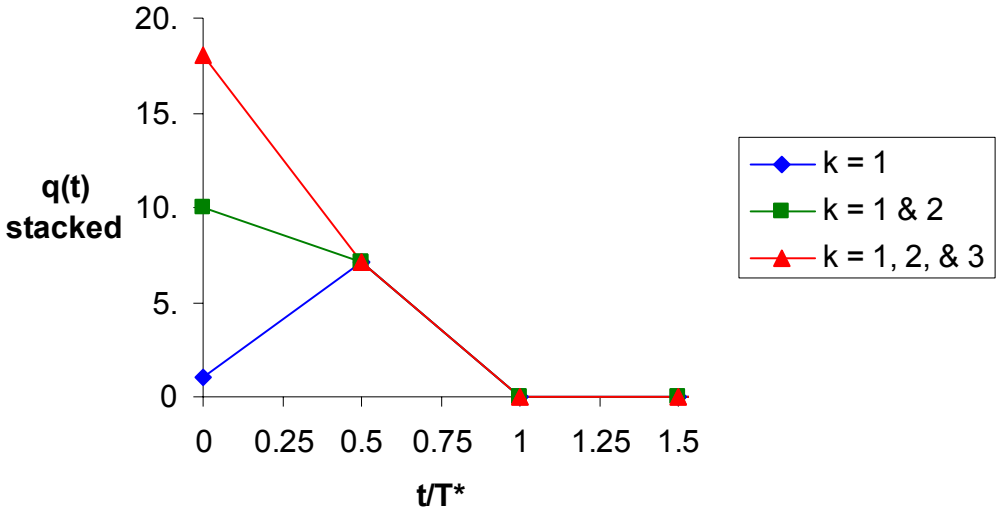

Figure 7.3: Example Total WIP in $\Delta t = T^*/2$ LP Optimal Solution

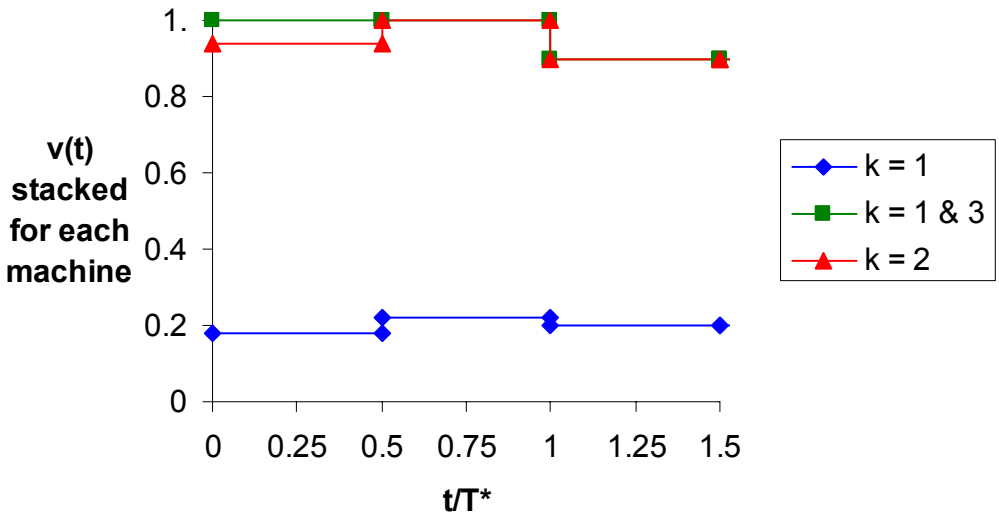

Figure 7.4: Example Machine Utilization in $\Delta t = T^*/2$ LP Optimal Solution

Figures 7.5 and 7.6 show the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ and the machine capacity allocation $\tilde{v}^{*}_{j,k}(t)$ over time for the initial feasible solution when Δt = T*/4. Figures 7.7 and 7.8 show the same for the optimal solution when Δt = T*/4 where the improvement in the objective function value is

$$\frac{\tilde{C}^{*}_{\mathrm{h}}(\tilde{\mathbf{Q}}^{+},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}},\tilde{\mathbf{Q}}\,|\,\Delta t = T^{*}/4,\ldots)}{\tilde{C}^{*}_{\mathrm{h}}(\tilde{\mathbf{Q}}^{+},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}},\tilde{\mathbf{Q}}\,|\,\Delta t = T^{*},\ldots)} = \frac{970.\overline{6}}{1152} \approx 0.8425925926\,.$$

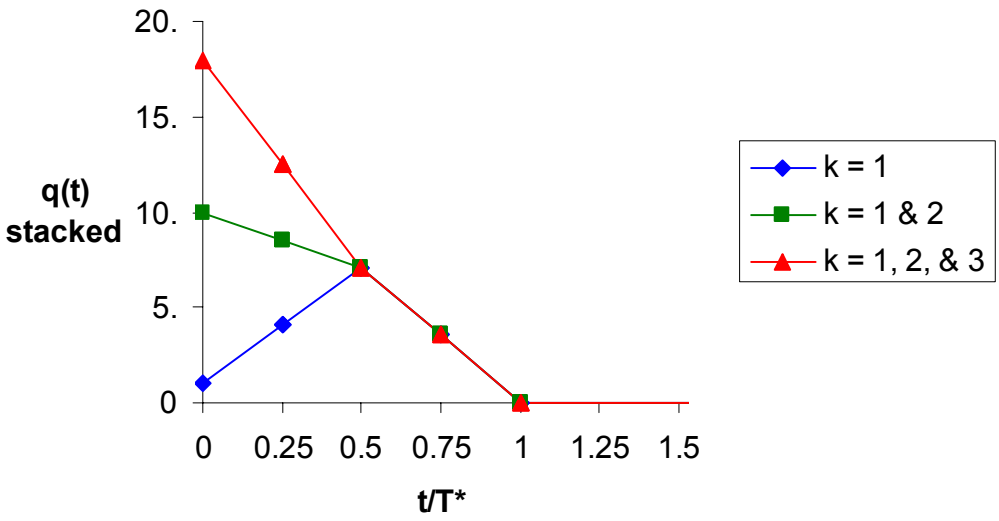

Figure 7.5: Example Total WIP in Δt = T*/4 LP Feasible Solution

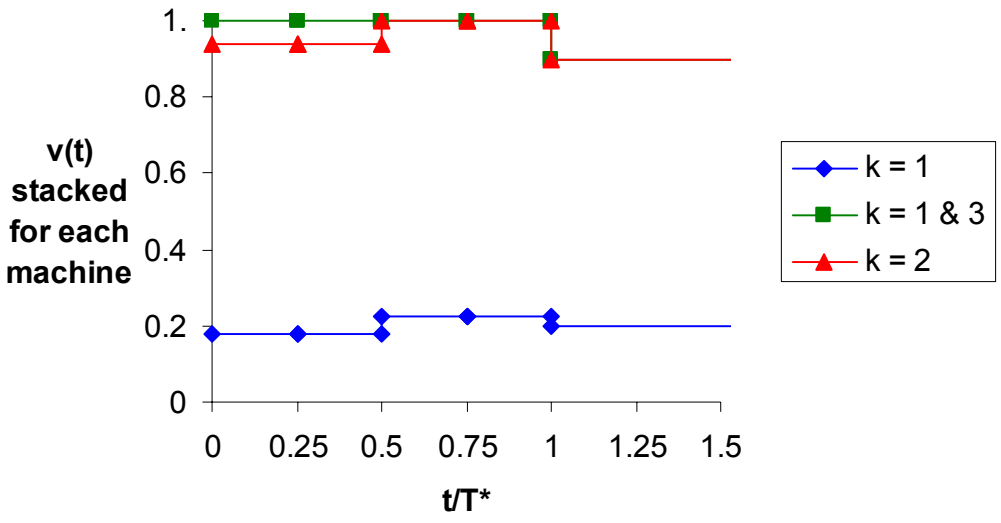

Figure 7.6: Example Machine Utilization in $\Delta t = T^*/4$ LP Feasible Solution

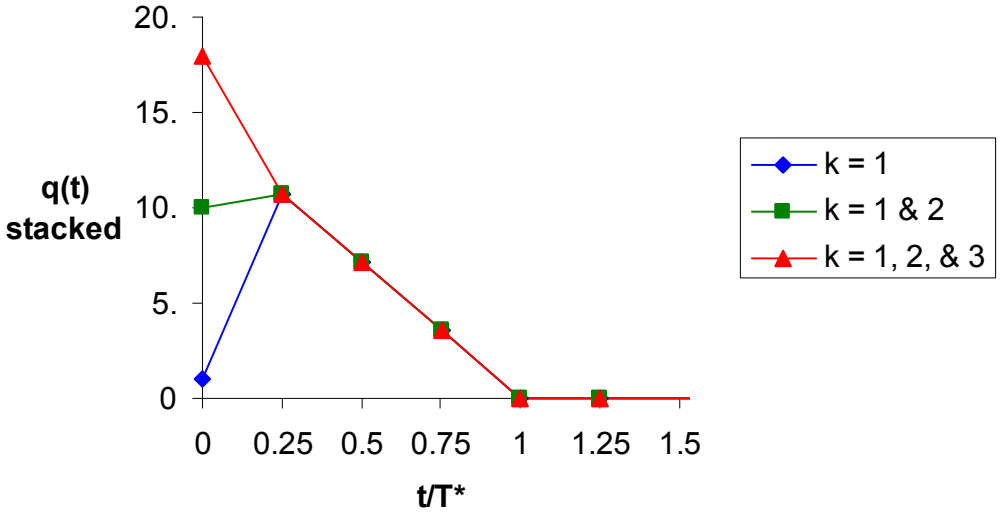

Figure 7.7: Example Total WIP in $\Delta t = T^*/4$ LP Optimal Solution

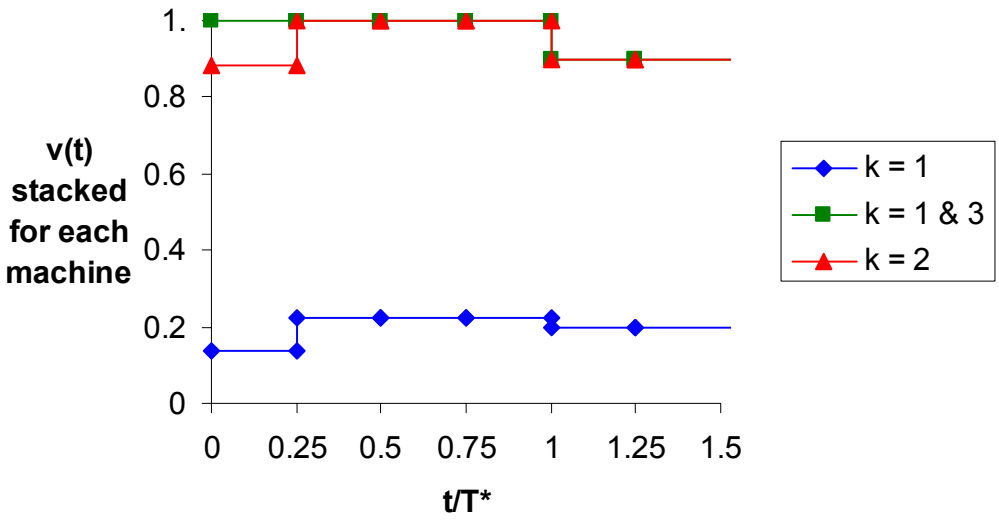

Figure 7.8: Example Machine Utilization in $\Delta t = T^*/4$ LP Optimal Solution

Figures 7.9 and 7.10 show the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ and the machine capacity allocation $\tilde{v}^{*}_{j,k}(t)$ over time for the initial feasible solution when $\Delta t = T^*/8$. Figures 7.11 and 7.12 show the same for the optimal solution when $\Delta t = T^*/8$ where the improvement in the objective function value is

$$\frac{\tilde{C}^{*}_{\mathrm{h}}(\tilde{\mathbf{Q}}^{+}, \Delta\tilde{\mathbf{U}}, \Delta\tilde{\mathbf{Y}}, \tilde{\mathbf{Q}} \mid \Delta t = T^*/8, \ldots)}{\tilde{C}^{*}_{\mathrm{h}}(\tilde{\mathbf{Q}}^{+}, \Delta\tilde{\mathbf{U}}, \Delta\tilde{\mathbf{Y}}, \tilde{\mathbf{Q}} \mid \Delta t = T^*, \ldots)} = \frac{940.\overline{4}}{1152} \approx 0.8163580247\,.$$

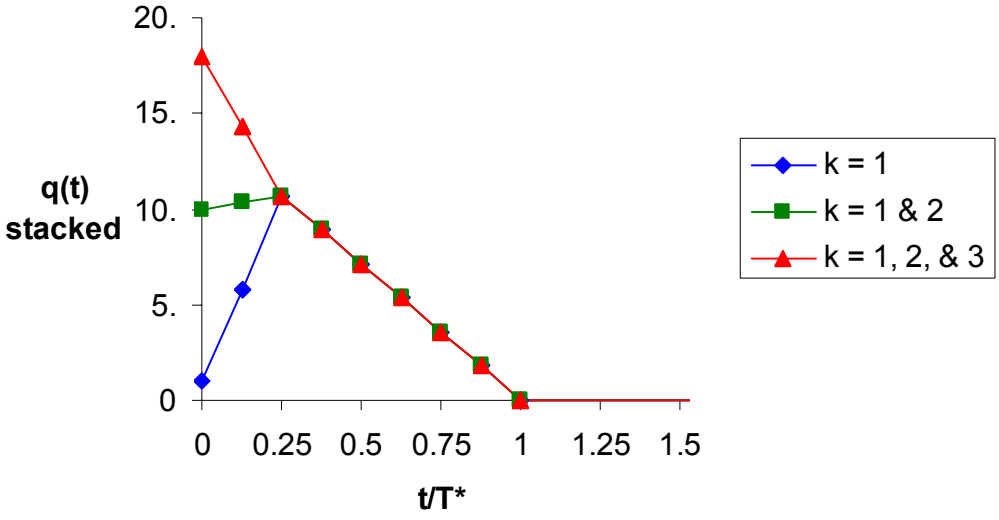

Figure 7.9: Example Total WIP in $\Delta t = T^*/8$ LP Feasible Solution

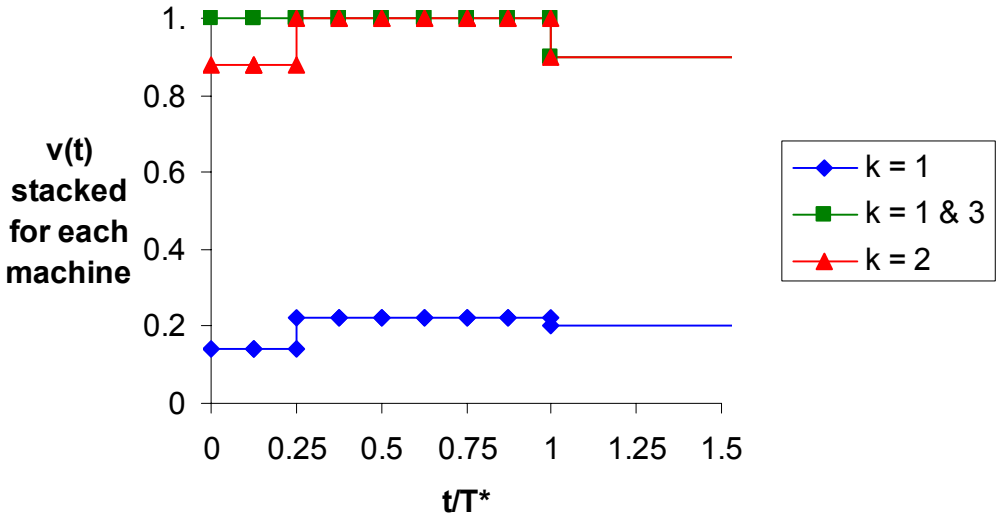

Figure 7.10: Example Machine Utilization in $\Delta t = T^*/8$ LP Feasible Solution

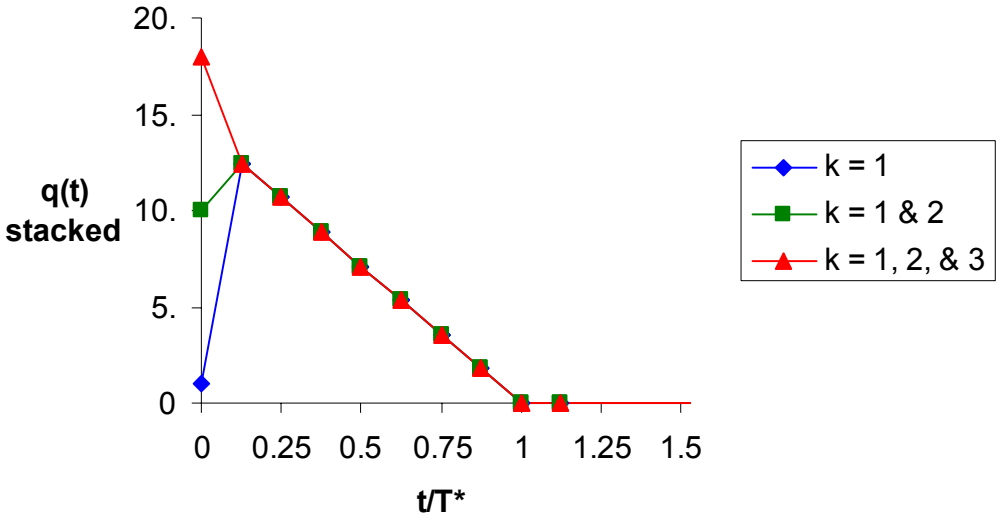

Figure 7.11: Example Total WIP in $\Delta t = T^*/8$ LP Optimal Solution

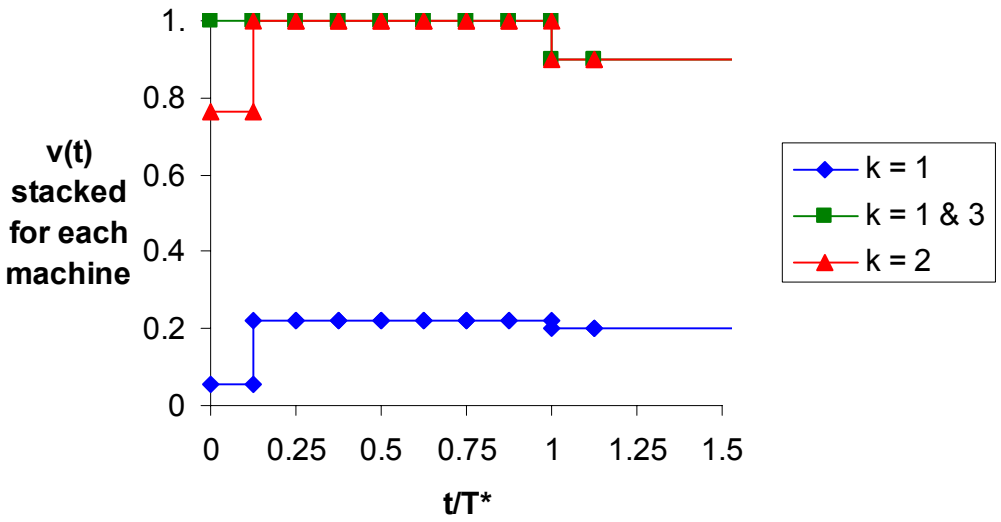

Figure 7.12: Example Machine Utilization in $\Delta t = T^*/8$ LP Optimal Solution

Figures 7.13 and 7.14 show the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ and the machine capacity allocation $\tilde{v}^{*}_{j,k}(t)$ over time for the initial feasible solution when Δt = T*/16. Figures 7.15 and 7.16 show the same for the optimal solution when Δt = T*/16 where the improvement in the objective function value is

$$\frac{\tilde{C}^{*}_{\mathrm{h}}(\tilde{\mathbf{Q}}^{+},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}},\tilde{\mathbf{Q}}\,|\,\Delta t = T^{*}/16,\ldots)}{\tilde{C}^{*}_{\mathrm{h}}(\tilde{\mathbf{Q}}^{+},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}},\tilde{\mathbf{Q}}\,|\,\Delta t = T^{*},\ldots)} \approx \frac{935.2381}{1152} \approx 0.8118386243.$$

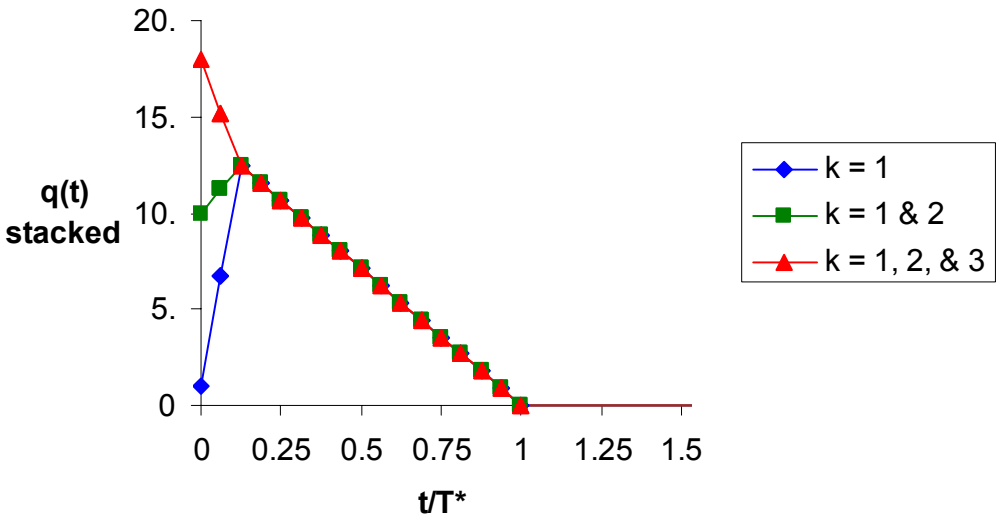

Figure 7.13: Example Total WIP in Δt = T*/16 LP Feasible Solution

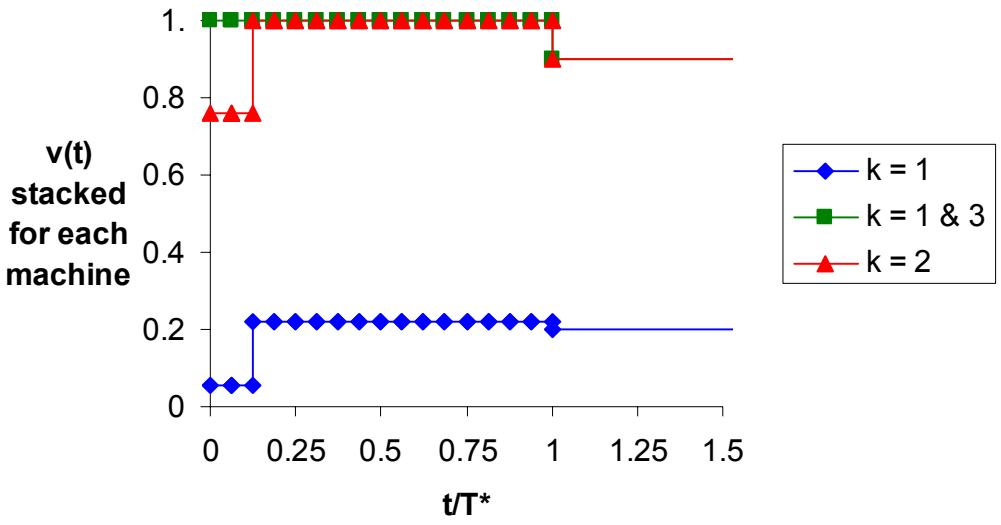

Figure 7.14: Example Machine Utilization in $\Delta t = T^*/16$ LP Feasible Solution

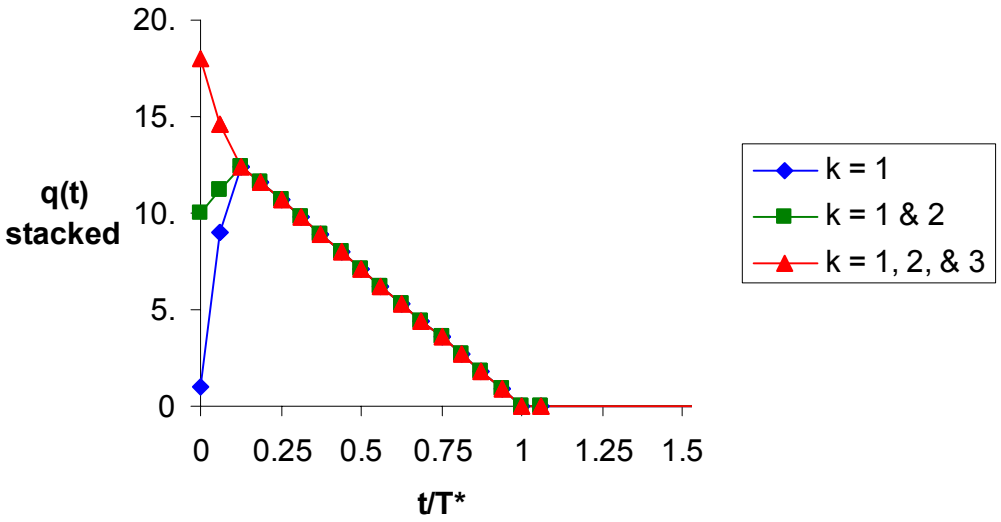

Figure 7.15: Example Total WIP in $\Delta t = T^*/16$ LP Optimal Solution

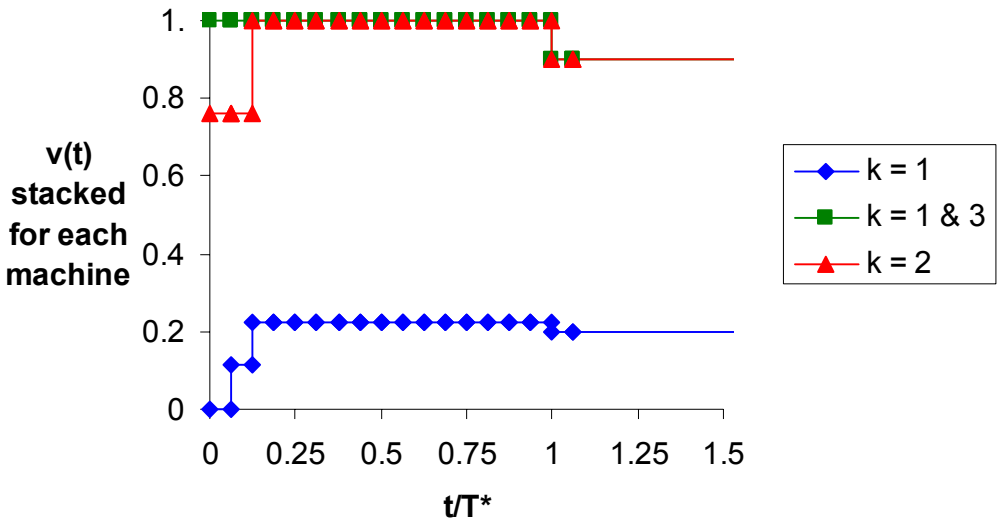

Figure 7.16: Example Machine Utilization in $\Delta t = T^*/16$ LP Optimal Solution

We can see from these figures that the sequence of LP optimal solutions do seem to be converging to an optimal solution to the CLP formulation. Pullan [1999] proposed more efficient (but more complicated) LP algorithms that allow time intervals of unequal lengths.

7.2 Linear Program (LP) for Fluid Relaxation with Fixed Total WIP Levels

On the other hand, if $\tilde{\mathbf{Q}}^+$ is held fixed (but feasible), then the vectors and matrices in equations (7.1) through (7.4) have the following form:

$$\mathbf{x} = \overset{\geq}{\mathbf{x}} = \Delta \mathbf{t}\,,\ \mathbf{f} = \mathbf{f}_{\Delta \mathbf{t}} + \mathbf{H}^{\mathrm{T}}_{\tilde{\mathbf{Q}}^+,\Delta \mathbf{t}} \mathbf{x}_{\tilde{\mathbf{Q}}^+}\,,\ c = 0\,,\ \overset{=}{\mathbf{A}} = \varnothing\,,\ \overset{=}{\mathbf{b}} = \varnothing\,,$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{\Delta\mathbf{U},\Delta\mathbf{t}} \\ \mathbf{A}_{\Delta\mathbf{Y},\Delta\mathbf{t}} \\ \mathbf{0}_{(N-1)K,N} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{b}_{\Delta\mathbf{U}} \\ \mathbf{b}_{\Delta\mathbf{Y}} \\ \mathbf{0}_{(N-1)K,1} \end{bmatrix} - \begin{bmatrix} \mathbf{A}_{\Delta\mathbf{U},\tilde{\mathbf{Q}}^+} \\ \mathbf{A}_{\Delta\mathbf{Y},\tilde{\mathbf{Q}}^+} \\ \mathbf{A}_{\mathbf{Q},\tilde{\mathbf{Q}}^+} \end{bmatrix} \mathbf{x}_{\tilde{\mathbf{Q}}^+},$$

where the submatrices are as they were defined in equations (6.9) through (6.14). Setting a matrix or vector equal to $\varnothing$ indicates that it has zero rows or columns (so it does not exist). Filling in the vectors and matrices gives Formulation 7.2:

Formulation 7.2: Linear Program (LP) for Fluid Relaxation with Fixed Total WIP Levels

$$\min_{\Delta\mathbf{t}} \tilde{C}_{\mathrm{h}}\left(\Delta\mathbf{t}, \Delta\tilde{\mathbf{U}}, \Delta\tilde{\mathbf{Y}}, \tilde{\mathbf{Q}} \mid N, \tilde{\mathbf{Q}}^+, \ldots\right)$$

$$= \frac{\left(\mathbf{c}^+\right)^{\mathrm{T}}}{2}\left\{\Delta t_1\left[\mathbf{q}^+(t_0) + \tilde{\mathbf{q}}^+(t_1)\right] + \sum_{n=2}^{N-1} \Delta t_n\left[\tilde{\mathbf{q}}^+(t_{n-1}) + \tilde{\mathbf{q}}^+(t_n)\right] + \Delta t_N \tilde{\mathbf{q}}^+(t_{N-1})\right\}$$

s.t.

$$-\boldsymbol{\alpha}^+\Delta t_1 + \Delta\tilde{\mathbf{u}}(t_1) = \mathbf{q}^+(t_0) - \tilde{\mathbf{q}}^+(t_1)$$

$$-\boldsymbol{\alpha}^+\Delta t_n + \Delta\tilde{\mathbf{u}}(t_n) = \tilde{\mathbf{q}}^+(t_{n-1}) - \tilde{\mathbf{q}}^+(t_n) \qquad \forall n \in \{2,3,\ldots,N-1\}$$

$$-\boldsymbol{\alpha}^+\Delta t_N + \Delta\tilde{\mathbf{u}}(t_N) = \tilde{\mathbf{q}}^+(t_{N-1})$$

$$-\boldsymbol{\chi}\Delta t_1 + \Delta\tilde{\mathbf{y}}(t_1) = -\mathbf{BD}(\mathbf{p})\left[\mathbf{q}^+(t_0) - \tilde{\mathbf{q}}^+(t_1)\right]$$

$$-\boldsymbol{\chi}\Delta t_n + \Delta\tilde{\mathbf{y}}(t_n) = -\mathbf{BD}(\mathbf{p})\left[\tilde{\mathbf{q}}^+(t_{n-1}) - \tilde{\mathbf{q}}^+(t_n)\right] \qquad \forall n \in \{2,3,\ldots,N-1\}$$

$$-\boldsymbol{\chi}\Delta t_N + \Delta\tilde{\mathbf{y}}(t_N) = -\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^+(t_{N-1})$$

$$\Delta t_n \geq 0 \qquad \forall n \in \{1,2,\ldots,N\}$$

$$\Delta\tilde{\mathbf{u}}(t_n) \geq \mathbf{0}_{K,1} \qquad \forall n \in \{1,2,\ldots,N\}$$

$$\Delta\tilde{\mathbf{y}}(t_n) \geq \mathbf{0}_{I,1} \qquad \forall n \in \{1,2,\ldots,N\}$$

Note that no constraint involves more than one decision variable. Thus, this LP formulation can be equivalently decomposed into N different LP formulations (one for each Δt_n) where the vectors and matrices in equations (7.1) through (7.4) have the following form $\forall n \in \{2,3,\ldots,N-1\}$:

$$\mathbf{x} = \overset{\geq}{\mathbf{x}} = \Delta t_n\,,\ \mathbf{f} = \frac{\left(\mathbf{c}^+\right)^{\mathrm{T}}}{2}\left[\tilde{\mathbf{q}}^+(t_{n-1}) + \tilde{\mathbf{q}}^+(t_n)\right],\ c = 0\,,$$

$$\mathbf{A} = \begin{bmatrix} -\boldsymbol{\alpha}^+ \\ -\boldsymbol{\chi} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \tilde{\mathbf{q}}^+(t_{n-1}) - \tilde{\mathbf{q}}^+(t_n) \\ -\mathbf{BD}(\mathbf{p})\left[\tilde{\mathbf{q}}^+(t_{n-1}) - \tilde{\mathbf{q}}^+(t_n)\right] \end{bmatrix}.$$

For $n = 1$, the vectors and matrices have the same form except that

$$\mathbf{f} = \frac{\left(\mathbf{c}^+\right)^{\mathbf{T}}}{2}\left[\mathbf{q}^+(t_0) + \tilde{\mathbf{q}}^+(t_1)\right] \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{q}^+(t_0) - \tilde{\mathbf{q}}^+(t_1) \\ -\mathbf{BD}(\mathbf{p})\left[\mathbf{q}^+(t_0) - \tilde{\mathbf{q}}^+(t_1)\right] \end{bmatrix}.$$

For $n = N$, the vectors and matrices have the same form except that

$$\mathbf{f} = \frac{\left(\mathbf{c}^+\right)^{\mathbf{T}}}{2}\tilde{\mathbf{q}}^+(t_{N-1}) \text{ and } \mathbf{b} = \begin{bmatrix} \tilde{\mathbf{q}}^+(t_{N-1}) \\ -\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^+(t_{N-1}) \end{bmatrix}.$$

Since $\mathbf{f}$ is nonnegative, we want Δt_n to be as small as possible, which is determined by the active constraint(s). If $\mathbf{P}$ is simple, we can see by inspection that the optimal solution is given by:

$$\Delta t_1^* = \max\left\{ \begin{array}{c} 0, \\ \max\limits_{\substack{j\in\{1,2,..,J\} \\ k\in\{1,2,..,k_j\}}} \dfrac{\tilde{q}^+_{j,k}(t_1) - q^+_{j,k}(t_0)}{\alpha^+_{j,k}}, \\ \max\limits_{i\in\{1,2,..,I\}} \dfrac{\sum\limits_{(j,k)\in\sigma_i} p_{j,k}\left[q^+_{j,k}(t_0) - \tilde{q}^+_{j,k}(t_1)\right]}{a_i M_i - \sum\limits_{(j,k)\in\sigma_i} p_{j,k}\alpha^+_{j,k}} \end{array} \right\},$$

$$\Delta t_N^* = \max\left\{ \begin{array}{c} 0, \\ \max\limits_{\substack{j\in\{1,2,\ldots,J\} \\ k\in\{1,2,\ldots,k_j\}}} \dfrac{-\tilde{q}_{j,k}^+(t_{N-1})}{\alpha_{j,k}^+}, \\ \max\limits_{i\in\{1,2,\ldots,I\}} \dfrac{\sum\limits_{(j,k)\in\sigma_i} p_{j,k}\tilde{q}_{j,k}^+(t_{N-1})}{a_i M_i - \sum\limits_{(j,k)\in\sigma_i} p_{j,k}\alpha_{j,k}^+} \end{array} \right\},$$

and $\forall n \in \{2,3,\ldots,N-1\}$

$$\Delta t_n^* = \max\left\{ \begin{array}{c} 0, \\ \max\limits_{\substack{j\in\{1,2,\ldots,J\} \\ k\in\{1,2,\ldots,k_j\}}} \dfrac{\tilde{q}_{j,k}^+(t_n) - \tilde{q}_{j,k}^+(t_{n-1})}{\alpha_{j,k}^+}, \\ \max\limits_{i\in\{1,2,\ldots,I\}} \dfrac{\sum\limits_{(j,k)\in\sigma_i} p_{j,k}\left[\tilde{q}_{j,k}^+(t_{n-1}) - \tilde{q}_{j,k}^+(t_n)\right]}{a_i M_i - \sum\limits_{(j,k)\in\sigma_i} p_{j,k}\alpha_{j,k}^+} \end{array} \right\}.$$

7.3 LINEAR PROGRAM (LP) FOR FLUID RELAXATION WITH ONE TIME INTERVAL

In the QP formulation, if $N = 1$, then $\mathbf{H} = \mathbf{0}$, $\underline{\tilde{\mathbf{Q}}} = \varnothing$, and there is no longer a quadratic term in the objective function, so we are left with Formulation 7.3, a one-dimensional linear program:

Formulation 7.3: Linear Program (LP) for Fluid Relaxation with One Time Interval

$$
\begin{aligned}
\min_{\Delta t_1} \tilde{C}_{\mathrm{h}}(\Delta t_1 \mid N=1,\ldots) &= \frac{\mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)}{2}\Delta t_1 \\
\text{s.t.} \quad & -\boldsymbol{\alpha}^{+}\Delta t_1 + \Delta\tilde{\mathbf{u}}(t_1) = \mathbf{q}^{+}(t_0) \\
& -\boldsymbol{\chi}\Delta t_1 + \Delta\tilde{\mathbf{y}}(t_1) = -\mathbf{B}\mathbf{D}(\mathbf{p})\mathbf{q}^{+}(t_0) \\
& \Delta t_1 \geq 0 \\
& \Delta\tilde{\mathbf{u}}(t_1) \geq \mathbf{0}_{K,1} \\
& \Delta\tilde{\mathbf{y}}(t_1) \geq \mathbf{0}_{I,1}
\end{aligned}
$$

This is identical to the previous LP formulation with fixed $\tilde{\mathbf{Q}}^{+}$ when $N = 1$. If $\mathbf{P}$ is simple, we can see by inspection that the optimal solution is given by:

$$
\begin{aligned}
\Delta t_1^{*} &= \max\left\{ \begin{array}{c} 0, \\ \max\limits_{\substack{j\in\{1,2,\ldots,J\} \\ k\in\{1,2,\ldots,k_j\}}} \dfrac{-q_{j,k}^{+}(t_0)}{\alpha_{j,k}^{+}}, \\ \max\limits_{i\in\{1,2,\ldots,I\}} \dfrac{\sum\limits_{(j,k)\in\sigma_i} p_{j,k}\tilde{q}_{j,k}^{+}(t_0)}{a_i M_i - \sum\limits_{(j,k)\in\sigma_i} p_{j,k}\alpha_{j,k}^{+}} \end{array} \right\} \\
&= \max\left\{ \frac{\sum\limits_{(j,k)\in\sigma_i} p_{j,k}q_{j,k}^{+}(t_0)}{a_i M_i - \sum\limits_{(j,k)\in\sigma_i} p_{j,k}\alpha_{j,k}^{+}} : i \in \{1,2,\ldots,I\} \right\} \\
&= T^{*},
\end{aligned}
$$

which is positive if the stability condition is met. The optimal objective function value is then

$$
\tilde{C}_{\mathrm{h}}^{*}(\Delta t_1 \mid N=1,\ldots) = \frac{\mathbf{c}^{\mathrm{T}}\mathbf{q}(t_0)}{2}T^{*}.
$$

Thus, the optimal solution to the weighted holding cost problem in the QP formulation for $N = 1$ is identical to the optimal makespan solution. For each $N \geq 1$, the optimal solution to the weighted holding cost problem in the QP formulation is a feasible solution for larger N. This suggests the following algorithm for finding the optimal solution to the weighted holding cost problem in the CLP formulation alluded to in the previous chapter:

1. Begin with $N = 1$ and the optimal makespan solution.
2. Increment N and solve the QP formulation with a new beginning feasible solution that is either an interpolated makespan solution or the prior optimal solution with a new break point inserted judiciously.
3. If the beginning feasible solution is still optimal for the larger N, stop. Otherwise, go to step 2.

Since N needs to be no larger than 2^{2K}, this is a finite quadratic programming algorithm for finding the optimal solution to the weighted holding cost problem in the CLP formulation. (In contrast, Luo and Bertsimas [1998] developed a convergent quadratic programming algorithm to accomplish the same result.)

However, since the QP formulation is not convex, solutions can only be guaranteed to be within a certain tolerance of the optimum. Thus, this finite quadratic programming algorithm still only gives approximately optimal solutions for the CLP formulation and is conceptually no better than the convergent QP and LP solution methods proposed by others. Also, as the number of breakpoints N approaches the number of queues K, the number of nonzero elements in the

coefficient matrix is of the order K^4, which for industrial sized problems may be computationally intractable or prohibitively expensive.

7.4 LINEAR PROGRAM (LP) FOR FLUID RELAXATION WITH ALL VARIABLES FIXED EXCEPT AT ONE TIME BREAK POINT

A new way (that others have not proposed) to get an LP formulation from the QP formulation is to fix all of the decision variables (at some set of feasible values), except for $\tilde{\mathbf{q}}^+(t_{n'})$, $\Delta t_{n'}$, and $\Delta t_{n'+1}$, but keeping $T' \equiv \Delta t_{n'} + \Delta t_{n'+1}$ fixed for some $n' \in \{1,2,...,N-1\}$. The QP formulation then reduces to Formulation 7.4:

Formulation 7.4: Linear Program (LP) for Fluid Relaxation with All Variables Fixed Except at One Time Break Point

$$\min_{\tilde{\mathbf{q}}^+(t_{n'}),\Delta t_{n'}} \tilde{C}_{\mathrm{h}}\left(\tilde{\mathbf{q}}^+(t_{n'}),\Delta t_{n'},\Delta\tilde{\mathbf{U}},\Delta\tilde{\mathbf{Y}},\tilde{\mathbf{Q}} \mid N,\left\{\tilde{\mathbf{q}}^+(t_n)\right\}_{n\neq n'},\left\{\Delta t_n\right\}_{n\neq n'},T',...\right)$$

$$= \frac{\left(\mathbf{c}^+\right)^{\mathrm{T}}}{2}\left[\mathbf{q}^+(t_0)\Delta t_1 + \sum_{n=1}^{N-1}\tilde{\mathbf{q}}^+(t_n)\left(\Delta t_n + \Delta t_{n+1}\right)\right]$$

s.t.
$$-\boldsymbol{\alpha}^+\Delta t_{n'} + \tilde{\mathbf{q}}^+(t_{n'}) + \Delta\tilde{\mathbf{u}}(t_{n'}) = \tilde{\mathbf{q}}^+(t_{n'-1})$$
$$\boldsymbol{\alpha}^+\Delta t_{n'} - \tilde{\mathbf{q}}^+(t_{n'}) + \Delta\tilde{\mathbf{u}}(t_{n'+1}) = \boldsymbol{\alpha}^+T' - \tilde{\mathbf{q}}^+(t_{n'+1})$$
$$-\boldsymbol{\chi}\Delta t_{n'} - \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^+(t_{n'}) + \Delta\tilde{\mathbf{y}}(t_{n'}) = -\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^+(t_{n'-1})$$
$$\boldsymbol{\chi}\Delta t_{n'} + \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^+(t_{n'}) + \Delta\tilde{\mathbf{y}}(t_{n'+1}) = \boldsymbol{\chi}T' + \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^+(t_{n'+1})$$
$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\tilde{\mathbf{q}}^+(t_{n'}) + \tilde{\mathbf{q}}(t_{n'}) = \mathbf{0}_{K,1}$$
$$0 \le \Delta t_{n'} \le T'$$
$$\tilde{\mathbf{q}}^+(t_{n'}) \in \mathbb{R}^K \left(\text{unrestricted}\right)$$
$$\tilde{\mathbf{q}}(t_{n'}) \ge \mathbf{0}_{K,1}$$
$$\Delta\tilde{\mathbf{u}}(t_{n'}) \ge \mathbf{0}_{K,1}$$
$$\Delta\tilde{\mathbf{u}}(t_{n'+1}) \ge \mathbf{0}_{K,1}$$
$$\Delta\tilde{\mathbf{y}}(t_{n'}) \ge \mathbf{0}_{I,1}$$
$$\Delta\tilde{\mathbf{y}}(t_{n'+1}) \ge \mathbf{0}_{I,1}$$

For this LP formulation, the vectors and matrices in equations (7.1) through (7.4) have the following form:

$$\mathbf{x} = \begin{bmatrix} \Delta t_{n'} \\ \tilde{\mathbf{q}}^{+}(t_{n'}) \end{bmatrix},\ \overset{\geq}{\mathbf{x}} = \Delta t_n,\ \mathbf{f} = \begin{bmatrix} \left[\tilde{\mathbf{q}}^{+}(t_{n'-1}) - \tilde{\mathbf{q}}^{+}(t_{n'+1})\right]^{\mathrm{T}} \mathbf{c}^{+}/2 \\ T'\mathbf{c}^{+}/2 \end{bmatrix},\ \overset{=}{\mathbf{A}} = \varnothing,\ \overset{=}{\mathbf{b}} = \varnothing,$$

$$c = \frac{\left(\mathbf{c}^{+}\right)^{\mathrm{T}}}{2}\Big[\mathbf{q}^{+}(t_0)\Delta t_1 \quad + \sum_{n=1}^{n'-2} \tilde{\mathbf{q}}^{+}(t_n)(\Delta t_n + \Delta t_{n+1}) + \tilde{\mathbf{q}}^{+}(t_{n'-1})\Delta t_{n'-1} + \tilde{\mathbf{q}}^{+}(t_{n'+1})(T' + \Delta t_{n'+2}) + \sum_{n=n'+2}^{N-1} \tilde{\mathbf{q}}^{+}(t_n)(\Delta t_n + \Delta t_{n+1})\Big],$$

$$\mathbf{A} = \begin{bmatrix} -\boldsymbol{\alpha}^{+} & \mathbf{I}_K \\ \boldsymbol{\alpha}^{+} & -\mathbf{I}_K \\ -\boldsymbol{\chi} & -\mathbf{BD}(\mathbf{p}) \\ \boldsymbol{\chi} & \mathbf{BD}(\mathbf{p}) \\ \mathbf{0}_{K,1} & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) \\ 1 & \mathbf{0}_{1,K} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \tilde{\mathbf{q}}^{+}(t_{n'-1}) \\ \boldsymbol{\alpha}^{+}T' - \tilde{\mathbf{q}}^{+}(t_{n'+1}) \\ -\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'-1}) \\ \boldsymbol{\chi}T' + \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'+1}) \\ \mathbf{0}_{K,1} \\ T' \end{bmatrix}.$$

However, most optimization software can handle the upper bound T' on $\Delta t_{n'}$ more elegantly than by adding a constraint.

This LP formulation is especially useful for inserting a new break point into an existing feasible solution and incrementing N. If we are interpolating $t_{n'}$ half-way between two adjacent break points, then

$$\Delta t_{n'} = \frac{T'}{2},$$

$$\tilde{\mathbf{q}}^{+}(t_{n'}) = \frac{\tilde{\mathbf{q}}^{+}(t_{n'-1}) + \mathbf{q}^{+}(t_{n'+1})}{2},$$

and

$$\mathbf{f}^{\mathrm{T}}\mathbf{x} = \frac{T'}{2}\left(\mathbf{c}^{+}\right)^{\mathrm{T}} \tilde{\mathbf{q}}^{+}(t_{n'-1}).$$

These vectors and matrices have the dimensions and sparsity shown in Table 7.2:

Table 7.2: Dimensions and Sparsity of Vectors and Matrices in LP Formulation with All Variables Fixed Except at One Time Break Point

Name	*Number of Rows/Columns*	*Number of Elements*	*Maximum Number of Nonzero Elements*
x	$K+1$ / 1	-	-
c	1/1	1	1
f	$K+1$ / 1	$K+1$	$K+1$ or, if $\mathbf{c}^{+}$ is simple, $J+1$
A	$3K+2I+1$ / $K+1$	$3K^2+2IK+4K+2I+1$	$K^2+6K+2I+1$ or, if **P** is simple, $8K+2I-J+1$
b	$3K+2I+1$ / 1	$3K+2I+1$	$2K+2I+1$
Total	-	$3K^2+2IK+8K+4I+4$	$K^2+9K+4I+4$ or, if **P** is simple, $11K+4I-J+4$ or, if $\mathbf{c}^{+}$ is too, $10K+4I+4$

Note that this formulation has two pairs of vector constraints that can be combined into the following two vector constraints with upper and lower bounds:

$$\tilde{\mathbf{q}}^{+}(t_{n'+1}) - \boldsymbol{\alpha}^{+}T' \le -\boldsymbol{\alpha}^{+}\Delta t_{n'} + \tilde{\mathbf{q}}^{+}(t_{n'}) \le \tilde{\mathbf{q}}^{+}(t_{n'-1}),$$

$$\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'-1}) \le \boldsymbol{\chi}\Delta t_{n'} + \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'}) \le \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'+1}) + \boldsymbol{\chi}T',$$

For optimization software (such as GAMS) that does not allow such "double-sided" constraints, we can introduce new decision variables with those upper and lower bounds:

$$-\boldsymbol{\alpha}^{+}\Delta t_{n'} + \tilde{\mathbf{q}}^{+}(t_{n'}) = \Delta\mathbf{u},$$

$$\tilde{\mathbf{q}}^{+}(t_{n'+1}) - \boldsymbol{\alpha}^{+}T' \le \Delta\mathbf{u} \le \tilde{\mathbf{q}}^{+}(t_{n'-1}),$$

$$\boldsymbol{\chi}\Delta t_{n'} + \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'}) = \Delta\mathbf{y},$$

$$\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'-1}) \le \Delta\mathbf{y} \le \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'+1}) + \boldsymbol{\chi}T',$$

which results in Formulation 7.5:

Formulation 7.5: Linear Program (LP) for Fluid Relaxation with All Variables Fixed Except at One Time Break Point (Alternate Form)

$$\min_{\tilde{\mathbf{q}}^{+}(t_{n'}),\Delta t_{n'},\Delta\mathbf{u},\Delta\mathbf{y}} \tilde{C}_{\mathrm{h}}\left(\tilde{\mathbf{q}}^{+}(t_{n'}),\Delta t_{n'},\Delta\mathbf{u},\Delta\mathbf{y},\tilde{\mathbf{q}}(t_{n'}) \mid N,\left\{\tilde{\mathbf{q}}^{+}(t_n)\right\}_{n\neq n'},\left\{\Delta t_n\right\}_{n\neq n'},T',\ldots\right)$$

$$= \frac{\left(\mathbf{c}^{+}\right)^{\mathrm{T}}}{2}\left[\mathbf{q}^{+}(t_0)\Delta t_1 + \sum_{n=1}^{N-1}\tilde{\mathbf{q}}^{+}(t_n)\left(\Delta t_n + \Delta t_{n+1}\right)\right]$$

s.t.
$$-\boldsymbol{\alpha}^{+}\Delta t_{n'} + \tilde{\mathbf{q}}^{+}(t_{n'}) - \Delta\mathbf{u} = \mathbf{0}_{K,1}$$
$$\boldsymbol{\chi}\Delta t_{n'} + \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'}) - \Delta\mathbf{y} = \mathbf{0}_{I,1}$$
$$-\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)\tilde{\mathbf{q}}^{+}(t_{n'}) + \tilde{\mathbf{q}}(t_{n'}) = \mathbf{0}_{K,1}$$
$$0 \le \Delta t_{n'} \le T'$$
$$\tilde{\mathbf{q}}^{+}(t_{n'+1}) - \boldsymbol{\alpha}^{+}T' \le \Delta\mathbf{u} \le \tilde{\mathbf{q}}^{+}(t_{n'-1})$$
$$\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'-1}) \le \Delta\mathbf{y} \le \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'+1}) + \boldsymbol{\chi}T'$$
$$\tilde{\mathbf{q}}^{+}(t_{n'}) \in \mathbb{R}^{K}\left(\text{unrestricted}\right)$$
$$\tilde{\mathbf{q}}(t_{n'}) \ge \mathbf{0}_{K,1}$$

For this LP formulation, the vectors and matrices in equations (7.1) through (7.4) have the following form:

$$\mathbf{x} = \begin{bmatrix} \Delta t_{n'} \\ \tilde{\mathbf{q}}^{+}(t_{n'}) \\ \Delta \mathbf{u} \\ \Delta \mathbf{y} \end{bmatrix}, \overset{\geq}{\mathbf{x}} = \Delta t_n, \ \mathbf{f} = \begin{bmatrix} \left[\tilde{\mathbf{q}}^{+}(t_{n'-1}) - \tilde{\mathbf{q}}^{+}(t_{n'+1})\right]^{\mathrm{T}} \mathbf{c}^{+}/2 \\ T'\mathbf{c}^{+}/2 \\ \mathbf{0}_{K,1} \\ \mathbf{0}_{I,1} \end{bmatrix},$$

$$c = \frac{\left(\mathbf{c}^{+}\right)^{\mathrm{T}}}{2}\Big[\mathbf{q}^{+}(t_0)\Delta t_1 \quad + \sum_{n=1}^{n'-2} \tilde{\mathbf{q}}^{+}(t_n)(\Delta t_n + \Delta t_{n+1}) + \tilde{\mathbf{q}}^{+}(t_{n'-1})\Delta t_{n'-1} + \tilde{\mathbf{q}}^{+}(t_{n'+1})(T' + \Delta t_{n'+2}) + \sum_{n=n'+2}^{N-1} \tilde{\mathbf{q}}^{+}(t_n)(\Delta t_n + \Delta t_{n+1})\Big],$$

$$\overset{=}{\mathbf{A}} = \begin{bmatrix} -\boldsymbol{\alpha}^{+} & \mathbf{I}_K & -\mathbf{I}_K & \mathbf{0}_{K,I} \\ \boldsymbol{\chi} & \mathbf{BD}(\mathbf{p}) & \mathbf{0}_{I,K} & -\mathbf{I}_I \end{bmatrix}, \ \overset{=}{\mathbf{b}} = \mathbf{0}_{K+I,1},$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}_{K,1} & -\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right) & \mathbf{0}_{K,K} & \mathbf{0}_{K,I} \\ 1 & \mathbf{0}_{1,K} & \mathbf{0}_{1,K} & \mathbf{0}_{1,I} \\ \mathbf{0}_{K,1} & \mathbf{0}_{K,K} & -\mathbf{I}_K & \mathbf{0}_{K,I} \\ \mathbf{0}_{K,1} & \mathbf{0}_{K,K} & \mathbf{I}_K & \mathbf{0}_{K,I} \\ \mathbf{0}_{I,1} & \mathbf{0}_{I,K} & \mathbf{0}_{I,K} & -\mathbf{I}_I \\ \mathbf{0}_{I,1} & \mathbf{0}_{I,K} & \mathbf{0}_{I,K} & \mathbf{I}_I \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{0}_{K,1} \\ T' \\ \boldsymbol{\alpha}^{+}T' - \tilde{\mathbf{q}}^{+}(t_{n'+1}) \\ \tilde{\mathbf{q}}^{+}(t_{n'-1}) \\ -\mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'-1}) \\ \boldsymbol{\chi}T' + \mathbf{BD}(\mathbf{p})\tilde{\mathbf{q}}^{+}(t_{n'+1}) \end{bmatrix}.$$

However, most optimization software can handle the bounds on $\Delta t_{n'}$, $\Delta \mathbf{u}$, and $\Delta \mathbf{y}$ more elegantly than by adding constraints.

These vectors and matrices have the dimensions and sparsity shown in Table 7.3:

Table 7.3: Dimensions and Sparsity of Vectors and Matrices in LP Formulation with All Variables Fixed Except at One Time Break Point (Alternate Form)

Name	*Number of Rows/Columns*	*Number of Elements*	*Maximum Number of Nonzero Elements*
x	$2K+I+1$ / 1	-	-
c	1/1	1	1
f	$2K+I+1$ / 1	$2K+I+1$	$K+1$ or, if $\mathbf{c}^+$ is simple, $J+1$
A and $\underline{=}$ **A**	$4K+3I+1$ / $2K+I+1$	$8K^2+3I^2+10IK+6K+4I+1$	$K^2+6K+4I+1$ or, if **P** is simple, $8K+4I-J+1$
b and $\underline{=}$ **b**	$4K+3I+1$ / 1	$4K+3I+1$	$2K+2I+1$
Total	-	$8K^2+3I^2+10IK+12K+8I+4$	$K^2+9K+6I+4$ or, if **P** is simple, $11K+6I-J+4$ or, if $\mathbf{c}^+$ is too, $10K+6I+4$

This suggests an algorithm for finding a nearly optimal solution to the weighted holding cost problem in the CLP formulation:

1. Begin with $N = 1$ and the optimal makespan solution.
2. Increment N and let $n = 0$.
3. Increment n and insert a new break point into the nth time interval in the existing feasible solution by interpolating $t_{n'}$ half-way between two adjacent break points.
4. Solve this LP formulation with all variables fixed except at the nth time break point.

5. If the LP formulation finds an improved solution, go to step 2.

Otherwise, if $n < N$, throw away the inserted break point and go to step 3.

Else, if $n = N$, stop.

The sequence of solutions from this LP formulation are plotted in the following figures for the example network. Figures 7.17 and 7.18 show the total WIP profile $\tilde{q}^{+*}_{j,k}(t)$ and the machine capacity allocation $\tilde{v}^{*}_{j,k}(t)$ over time for the initial feasible solution when $N = 2$. Figures 7.19 and 7.20 show the same for the optimal solution when $N = 2$. This optimal solution is the same as for $N = 2$ in the QP optimal solution. Inserting a new break point into either of the resulting time intervals does not lead to an improved solution.

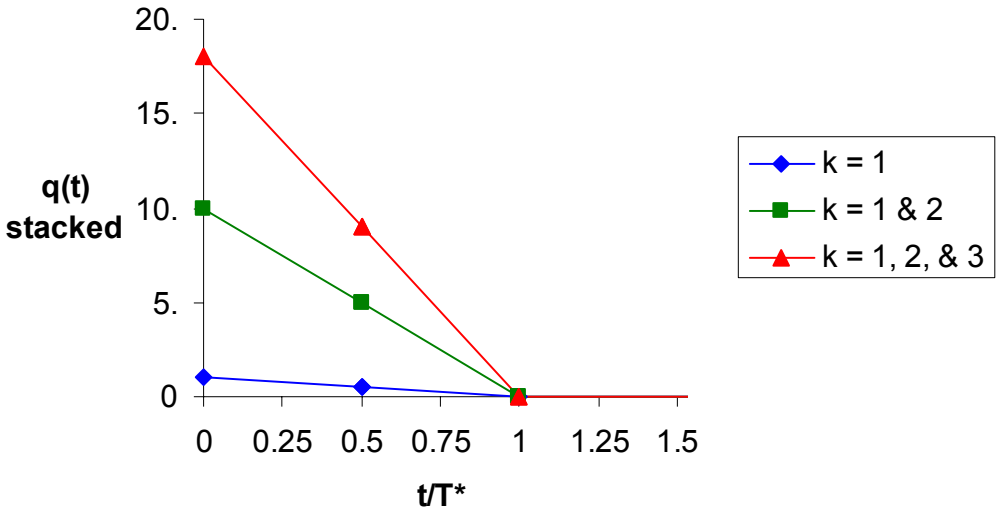

Figure 7.17: Example Total WIP in $N = 2$ LP Feasible Solution

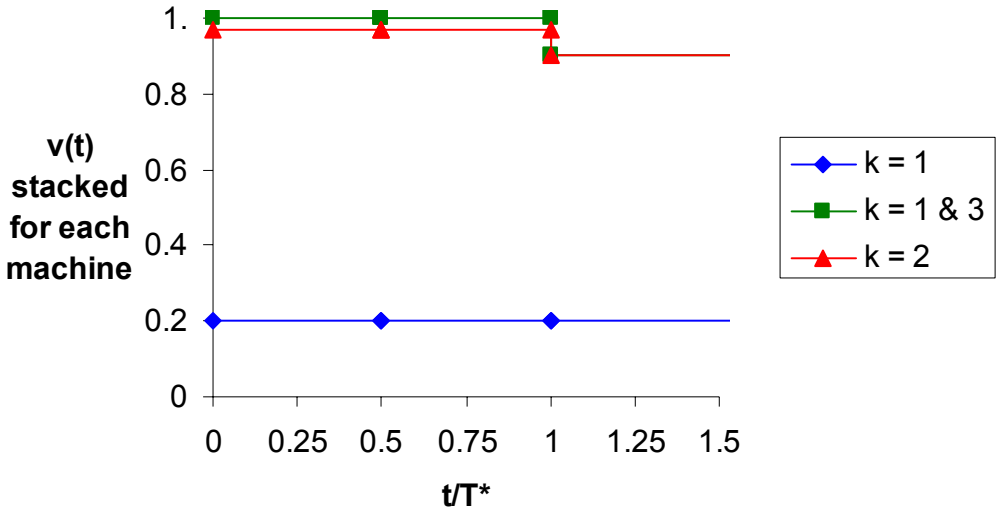

Figure 7.18: Example Machine Utilization in $N = 2$ LP Feasible Solution

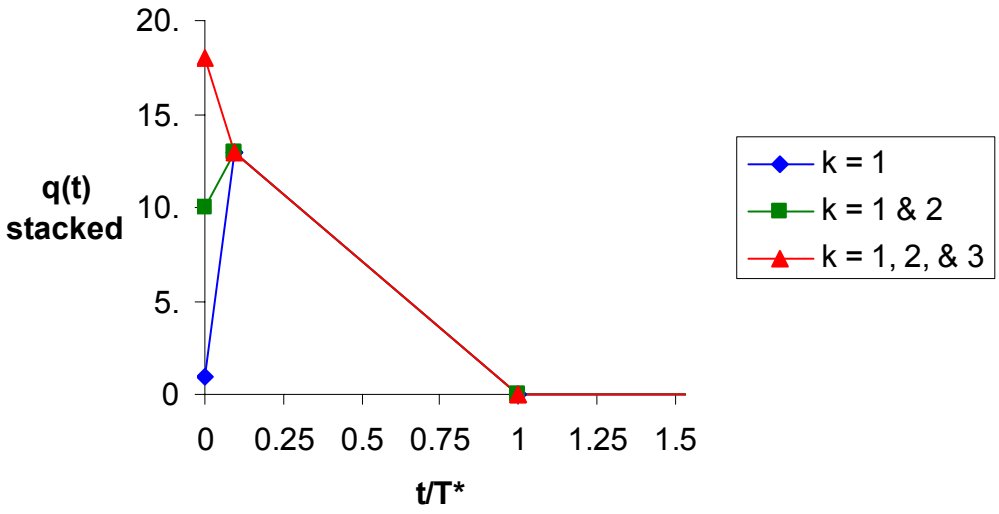

Figure 7.19: Example Total WIP in $N = 2$ LP Optimal Solution

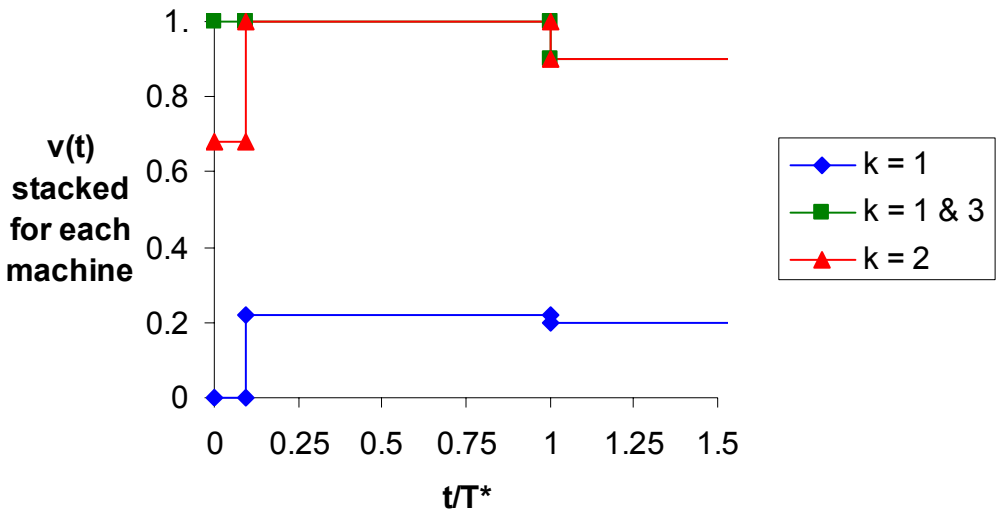

Figure 7.20: Example Machine Utilization in $N = 2$ LP Optimal Solution

Since the number of nonzero elements in the coefficient matrix remains only of the order K^2 no matter how many breakpoints are used, this small heuristic LP algorithm is easily implemented and is computationally very fast. However, we have no rigorous bounds on how close the resulting solution is to the optimal solution to the weighted holding cost problem in the CLP formulation. In practice, though, this heuristic LP algorithm tends to give solutions that are practically indistinguishable from optimal.

8. Mixed Integer Programming (MIP) Models: Schedule Initialization

In this chapter, we describe methods for initially allocating jobs to machines. As will be described in more detail in Chapter 9, one heuristic scheduling method attempts to keep $\hat{u}_{j,k}(t)$, the cumulative number of units that have left each queue over time in the discrete schedule, close to $\tilde{u}^*_{j,k}(t)$, the same variable in the optimal fluid schedule. However, these two variables should be identical at the beginning:

$$\hat{u}_{j,k}(t_0) = u_{j,k}(t_0) = \tilde{u}^*_{j,k}(t_0). \qquad \begin{cases} \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}$$

Thus, some other method is needed to initially allocate jobs to machines. An obvious method is to set $\hat{v}_{j,k}(t_0^+)$, the integer number of machines working on jobs of type *j* in stage *k* just after time t_0, to be as close as possible to $\tilde{v}^*_{j,k}(t_1)$, the same variable (but possibly not an integer) for the first time interval in the optimal fluid schedule. Note that although $\tilde{v}^*_{j,k}(t_1)$ was derived from a decision variable in the optimal fluid schedule, it is now just input data. This objective can be expressed as a minimization of some norm:

$$\min_{\hat{\mathbf{v}}(t_0^+)} \left\| \hat{\mathbf{v}}(t_0^+) - \tilde{\mathbf{v}}^*(t_1) \right\|. \tag{8.1}$$

However, several constraints need to be taken into consideration. First, no machine can process negative jobs or more jobs than are available, so

$$0 \le \hat{v}_{j,k}(t_0^+) \le q_{j,k}(t_0). \qquad \begin{cases} \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}$$

Thus,

$$\hat{v}_{j,k}(t_0^+) \in \left\{0,1,2,\ldots,q_{j,k}(t_0)\right\}. \qquad \begin{cases} \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}$$

Second, the number of jobs being processed cannot exceed the number of machines, so

$$\sum_{(j,k)\in\sigma_i} \hat{v}_{j,k}(t_0^+) \le \hat{M}_i \qquad \forall i \in \{1,2,...,I\}$$

where $\hat{M}_i$ is the actual number of machines of type i that are available (idle or processing, but not broken down) at time t_0. If a non-idling policy is in force, this inequality is replaced by the following equation:

$$\sum_{(j,k)\in\sigma_i} \hat{v}_{j,k}(t_0^+) = \min\left\{\hat{M}_i, \sum_{(j,k)\in\sigma_i} q_{j,k}(t_0)\right\}. \qquad \forall i \in \{1,2,...,I\}$$

Putting these equations and inequalities together gives Formulation 8.1:

Formulation 8.1: Mixed Integer Math Program (MIMP) for Schedule Initialization

$$\min_{\hat{\mathbf{v}}(t_0^+)} \left\| \hat{\mathbf{v}}(t_0^+) - \tilde{\mathbf{v}}^*(t_1) \right\|$$

$$\text{s.t.} \quad \mathbf{B}\hat{\mathbf{v}}(t_0^+) \begin{cases} \le \hat{\mathbf{m}} & \text{if idling is allowed} \\ \text{or} & \\ = \min\left\{\hat{\mathbf{m}}, \mathbf{B}\mathbf{q}(t_0)\right\} & \text{if idling is not allowed} \end{cases}$$

$$\hat{\mathbf{v}}(t_0^+) \begin{cases} \le \mathbf{q}(t_0) \\ \text{and} \\ \in \mathbb{Z}_+^K \end{cases}$$

For most choices of the norm, this formulation can be equivalently decomposed into I different MIMP formulations (one for each machine type).

8.1 MIXED INTEGER LINEAR PROGRAM (MILP) FOR l_1-NORM SCHEDULE INITIALIZATION

If the norm in objective (8.1) is the l_1 norm:

$$\left\|\mathbf{v}\right\|_1 \equiv \sum_{k=1}^{K} \left|v_k\right|,$$

then we introduce a new decision variable $\breve{\mathbf{v}} \in \mathbb{R}^K$ (the deviation from integrality in the number of machines working) into the objective function (8.1) to get:

$$\min_{\breve{\mathbf{v}}} \left\|\breve{\mathbf{v}}\right\|_1,$$

along with two new sets of constraints:

$$\breve{v}_{j,k} \geq \hat{v}_{j,k}(t_0^+) - \tilde{v}^*_{j,k}(t_1) \qquad \begin{cases} \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}$$

and

$$\breve{v}_{j,k} \geq \tilde{v}^*_{j,k}(t_1) - \hat{v}_{j,k}(t_0^+). \qquad \begin{cases} \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}$$

Since we are minimizing $\breve{\mathbf{v}}$, which must be positive, the result is that

$$\breve{v}_{j,k} = \left|\hat{v}_{j,k}(t_0^+) - \tilde{v}^*_{j,k}(t_1)\right| \qquad \begin{cases} \forall j \in \{1,2,...,J\} \\ \forall k \in \{2,3,...,K_j\} \end{cases}$$

and

$$\left\|\breve{\mathbf{v}}\right\|_1 = \mathbf{1}_K^{\mathrm{T}} \breve{\mathbf{v}}.$$

This allows us to equivalently express the schedule initialization problem as shown in Formulation 8.2:

Formulation 8.2: Mixed Integer Linear Program (MILP) for l_1-Norm Schedule Initialization

$$
\begin{aligned}
&\min_{\breve{\mathbf{v}},\hat{\mathbf{v}}(t_0^+)} \mathbf{1}_K^{\mathrm{T}} \breve{\mathbf{v}} \\
&\text{s.t.} \quad -\breve{\mathbf{v}} + \hat{\mathbf{v}}(t_0^+) \le \tilde{\mathbf{v}}^*(t_1) \\
&\qquad -\breve{\mathbf{v}} - \hat{\mathbf{v}}(t_0^+) \le -\tilde{\mathbf{v}}^*(t_1) \\
&\qquad \mathbf{B}\hat{\mathbf{v}}(t_0^+) \begin{cases} \le \hat{\mathbf{m}} & \text{if idling is allowed} \\ \text{or} & \\ = \min\{\hat{\mathbf{m}}, \mathbf{B}\mathbf{q}(t_0)\} & \text{if idling is not allowed} \end{cases} \\
&\qquad \hat{\mathbf{v}}(t_0^+) \begin{cases} \le \mathbf{q}(t_0) & \\ \text{and} & \\ \in \mathbb{Z}_+^K & \end{cases} \\
&\qquad \breve{\mathbf{v}} \in \mathbb{R}^K \text{ (unrestricted)}
\end{aligned}
$$

Since the objective function and the constraints are linear in the decision variables, some of which are restricted to be integers, this is a mixed-integer linear program (often just called a mixed-integer program) of the form

$$\min_{\mathbf{x}} \left(\mathbf{f}^{\mathrm{T}} \mathbf{x} + c \right) \tag{8.2}$$

$$\text{s.t.} \quad \overset{=}{\mathbf{A}}\,\mathbf{x} = \overset{=}{\mathbf{b}} \tag{8.3}$$

$$\mathbf{A}\mathbf{x} \le \mathbf{b} \tag{8.4}$$

$$\overset{\ge}{\mathbf{x}} \ge \mathbf{0} \tag{8.5}$$

$$\hat{\mathbf{x}} \text{ integer}. \tag{8.6}$$

The vectors and matrices in equations (8.2) through (8.6) have the following form:

$$\mathbf{x} = \begin{bmatrix} \hat{\mathbf{v}}(t_0^+) \\ \breve{\mathbf{v}} \end{bmatrix},\ \overset{\geq}{\mathbf{x}} = \hat{\mathbf{x}} = \hat{\mathbf{v}}(t_0^+),\ \mathbf{f} = \begin{bmatrix} \mathbf{0}_{K,1} \\ \mathbf{1}_K \end{bmatrix},\ c = 0. \tag{8.7}$$

If idling is allowed, there are no equality constraints (so $\overset{=}{\mathbf{A}}$ and $\overset{=}{\mathbf{b}}$ do not exist), and

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_K & -\mathbf{I}_K \\ -\mathbf{I}_K & -\mathbf{I}_K \\ \mathbf{I}_K & \mathbf{0}_{K,K} \\ \mathbf{B} & \mathbf{0}_{I,K} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \tilde{\mathbf{v}}^*(t_1) \\ -\tilde{\mathbf{v}}^*(t_1) \\ \mathbf{q}(t_0) \\ \hat{\mathbf{m}} \end{bmatrix}. \tag{8.8}$$

If idling is not allowed,

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_K & -\mathbf{I}_K \\ -\mathbf{I}_K & -\mathbf{I}_K \\ \mathbf{I}_K & \mathbf{0}_{K,K} \end{bmatrix},\ \mathbf{b} = \begin{bmatrix} \tilde{\mathbf{v}}^*(t_1) \\ -\tilde{\mathbf{v}}^*(t_1) \\ \mathbf{q}(t_0) \end{bmatrix}, \tag{8.9}$$

$$\overset{=}{\mathbf{A}} = \begin{bmatrix} \mathbf{B} & \mathbf{0}_{I,K} \end{bmatrix}, \text{ and } \overset{=}{\mathbf{b}} = \min\left\{\hat{\mathbf{m}}, \mathbf{B}\mathbf{q}(t_0)\right\}. \tag{8.10}$$

However, most optimization software can handle the upper bound $\mathbf{q}(t_0)$ on $\hat{\mathbf{v}}(t_0^+)$ more elegantly than by adding a constraint.

8.2 MIXED INTEGER LINEAR PROGRAM (MILP) FOR l_∞-NORM SCHEDULE INITIALIZATION

If the norm in objective (8.1) is the l_∞ norm:

$$\left\|\mathbf{v}\right\|_\infty \equiv \max_k \left\{\left|v_k\right|\right\},$$

then we introduce a new decision variable $\breve{\mathbf{v}} \in \mathbb{R}^I$ (instead of $\mathbb{R}^K$) into the objective function (8.1) to get:

$$\min_{\breve{\mathbf{v}}} \left\|\breve{\mathbf{v}}\right\|_\infty,$$

along with two new sets of constraints:

$$\breve{v}_i \geq \hat{v}_{j,k}(t_0^+) - \tilde{v}^*_{j,k}(t_1) \qquad \begin{cases} \forall i \in \{1,2,...,I\} \\ \forall (j,k) \in \sigma_i \end{cases}$$

and

$$\breve{v}_i \geq \tilde{v}^*_{j,k}(t_1) - \hat{v}_{j,k}(t_0^+). \qquad \begin{cases} \forall i \in \{1,2,...,I\} \\ \forall (j,k) \in \sigma_i \end{cases}$$

Since we are minimizing $\breve{\mathbf{v}}$, which must be positive, the result is that

$$\breve{v}_i = \max_{(j,k)\in\sigma_i} \left\{\left|\hat{v}_{j,k}(t_0^+) - \tilde{v}^*_{j,k}(t_1)\right|\right\} \qquad \forall i \in \{1,2,...,I\}$$

and again

$$\left\|\breve{\mathbf{v}}\right\|_\infty = \mathbf{1}_I^{\mathrm{T}} \breve{\mathbf{v}}.$$

This allows us to equivalently express the schedule initialization problem in another mixed integer linear programming formulation:

Formulation 8.3: Mixed Integer Linear Program (MILP) for l_∞-Norm Schedule Initialization

$$
\begin{aligned}
&\min_{\breve{\mathbf{v}},\hat{\mathbf{v}}(t_0^+)} \mathbf{1}_I^{\mathrm{T}} \breve{\mathbf{v}} \\
&\text{s.t.} \quad -\mathbf{B}^{\mathrm{T}} \breve{\mathbf{v}} + \hat{\mathbf{v}}(t_0^+) \le \tilde{\mathbf{v}}^*(t_1) \\
&\qquad -\mathbf{B}^{\mathrm{T}} \breve{\mathbf{v}} - \hat{\mathbf{v}}(t_0^+) \le -\tilde{\mathbf{v}}^*(t_1) \\
&\qquad \mathbf{B}\hat{\mathbf{v}}(t_0^+) \begin{cases} \le \hat{\mathbf{m}} & \text{if idling is allowed} \\ \text{or} & \\ = \min\{\hat{\mathbf{m}}, \mathbf{B}\mathbf{q}(t_0)\} & \text{if idling is not allowed} \end{cases} \\
&\qquad \hat{\mathbf{v}}(t_0^+) \begin{cases} \le \mathbf{q}(t_0) & \\ \text{and} & \\ \in \mathbb{Z}_+^K & \end{cases} \\
&\qquad \breve{\mathbf{v}} \in \mathbb{R}^I \left(\text{unrestricted}\right)
\end{aligned}
$$

The vectors and matrices in equations (8.2) through (8.6) have the same form as in equations (8.7) through (8.10) except that if idling is allowed,

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_K & -\mathbf{B}^{\mathrm{T}} \\ -\mathbf{I}_K & -\mathbf{B}^{\mathrm{T}} \\ \mathbf{I}_K & \mathbf{0}_{K,I} \\ \mathbf{B} & \mathbf{0}_{I,I} \end{bmatrix}.$$

If idling is not allowed,

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_K & -\mathbf{B}^{\mathrm{T}} \\ -\mathbf{I}_K & -\mathbf{B}^{\mathrm{T}} \\ \mathbf{I}_K & \mathbf{0}_{K,I} \end{bmatrix}, \text{ and } \bar{\bar{\mathbf{A}}} = \begin{bmatrix} \mathbf{B} & \mathbf{0}_{I,I} \end{bmatrix}.$$

8.3 MIXED INTEGER QUADRATIC PROGRAM (MIQP) FOR l_2 -NORM SCHEDULE INITIALIZATION

If the norm in objective (8.1) is the l_2 norm:

$$\|\mathbf{v}\|_2 \equiv \sqrt{\sum_{k=1}^{K} |v_k|^2},$$

then it is equivalent to minimize

$$\|\mathbf{v}\|_2^2 \equiv \sum_{k=1}^{K} |v_k|^2,$$

so an equivalent objective to (8.1) is

$$\min_{\hat{\mathbf{v}}(t_0^+)} \left[\hat{\mathbf{v}}(t_0^+)^{\mathbf{T}} \hat{\mathbf{v}}(t_0^+) - 2\tilde{\mathbf{v}}^*(t_1)^{\mathbf{T}} \hat{\mathbf{v}}(t_0^+) + \tilde{\mathbf{v}}^*(t_1)^{\mathbf{T}} \tilde{\mathbf{v}}^*(t_1) \right].$$

Dropping the constant $\tilde{\mathbf{v}}^*(t_1)^{\mathbf{T}} \tilde{\mathbf{v}}^*(t_1)$ term, we can express the schedule initialization problem as shown in Formulation 8.4:

Formulation 8.4: Mixed Integer Quadratic Program (MIQP) for l_2 -Norm Schedule Initialization

$$\min_{\hat{\mathbf{v}}(t_0^+)} \left[\hat{\mathbf{v}}(t_0^+)^{\mathbf{T}} \hat{\mathbf{v}}(t_0^+) - 2\tilde{\mathbf{v}}^*(t_1)^{\mathbf{T}} \hat{\mathbf{v}}(t_0^+) \right]$$

$$\text{s.t.} \quad \mathbf{B}\hat{\mathbf{v}}(t_0^+) \begin{cases} \leq \hat{\mathbf{m}} & \text{if idling is allowed} \\ \text{or} & \\ = \min\{\hat{\mathbf{m}}, \mathbf{B}\mathbf{q}(t_0)\} & \text{if idling is not allowed} \end{cases}$$

$$\hat{\mathbf{v}}(t_0^+) \begin{cases} \leq \mathbf{q}(t_0) \\ \text{and} \\ \in \mathbb{Z}_+^K \end{cases}$$

Since the constraints are linear and the objective function is quadratic in the decision variables, some of which are restricted to be integers, this is a mixed integer quadratic program of the form

$$\min_{\mathbf{x}} \left(\frac{\mathbf{x}^{\mathrm{T}}\mathbf{H}\mathbf{x}}{2} + \mathbf{f}^{\mathrm{T}}\mathbf{x} \right) \tag{8.11}$$

$$\text{s.t.} \quad \overset{=}{\mathbf{A}}\,\mathbf{x} = \overset{=}{\mathbf{b}} \tag{8.12}$$

$$\mathbf{A}\mathbf{x} \le \mathbf{b} \tag{8.13}$$

$$\overset{\geq}{\mathbf{x}} \ge \mathbf{0} \tag{8.14}$$

$$\hat{\mathbf{x}} \text{ integer}. \tag{8.15}$$

The vectors and matrices in equations (8.11) through (8.15) have the following form:

$\mathbf{x} = \overset{\geq}{\mathbf{x}} = \hat{\mathbf{x}} = \hat{\mathbf{v}}(t_0^+)$, $\mathbf{f} = -2\tilde{\mathbf{v}}^*(t_1)$, $\mathbf{H} = 2\mathbf{I}_K$.

If idling is allowed, there are no equality constraints (so $\overset{=}{\mathbf{A}}$ and $\overset{=}{\mathbf{b}}$ do not exist), and

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{B} \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{q}(t_0) \\ \hat{\mathbf{m}} \end{bmatrix}.$$

If idling is not allowed,

$\mathbf{A} = \mathbf{I}_K$, $\mathbf{b} = \mathbf{q}(t_0)$, $\overset{=}{\mathbf{A}} = \mathbf{B}$, and $\overset{=}{\mathbf{b}} = \min\{\hat{\mathbf{m}}, \mathbf{B}\mathbf{q}(t_0)\}$.

Again, most optimization software can handle the upper bound $\mathbf{q}(t_0)$ on $\hat{\mathbf{v}}(t_0^+)$ more elegantly than by adding a constraint.

9. Discrete-Event Simulation Models: Stochastic Job Shop

To test the ideas in this document, a discrete-event software simulation of a factory was written in the Sigma™ resource-based simulation environment developed by Schruben [1997]. This simulation incorporates all of the features specified in Feigin et al. [1994] with the exception of operators and mixed batches. The simulation does allow any number of buffers and machine types, and the time between arrivals, processing times, and times to transfer between stages can all be deterministic or exponentially distributed (with independent random number streams for each arrival). Each machine type can have up to five failure modes (either based on processing time or number of processing cycles) each with its own mean time (or cycles) to fail (MTTF) and mean time to repair (MTTR). The time to repair is exponentially distributed and either the time to fail is exponentially distributed or the number or runs between failures is geometrically distributed. Only full batches have been implemented (the possibility of running partial and mixed batches should be addressed in the future). Sequence-dependent setup times and probabilistic routing are also implemented.

A variety of scheduling and dispatching policies have been implemented in the simulation using the input parameter `Policy`. If `Policy` has a value of one through four, then the buffers are respectively served in first-in-first-out (FIFO), last-in-first-out (LIFO), first-buffer-first-served (FBFS), and last-buffer-first-served (LBFS) order. If `Policy` has a value of five, then the buffers are served in the priority order given in the input file (but with a minimum run length also specified in the input file). If `Policy` has a value of six (which only applies

to systems with three buffers like the example network), then which buffer is served depends on which side of a planar boundary **q** falls in $\mathbb{Z}_+^3$ (as described at the end of Chapter 3) .

If `Policy` has a value of zero, then the simulation attempts to follow a fluid solution given in the input file. A variety of methods exist for adapting the optimal fluid solution to make a feasible discrete factory schedule. The simplest way is to have a dispatching rule such that when a machine becomes available, it next runs jobs from whichever queue has fallen most behind the fluid solution. This is queue (j,k) for which $\tilde{u}_{j,k}(t) - \hat{u}_{j,k}(t)$ is largest, with ties going to the queue that was just served by the machine (if it was tied for being most behind) or arbitrarily (otherwise). Batching was easily implemented, since any discrete batch size is qualitatively a comparable departure from the continuous fluid solution.

One problem with this naive approach is that it may cause machines to switch frequently between processing jobs from different queues. If there are sequence-dependent setup times (which the fluid model does not explicitly comprehend), this can significantly slow down the system. One way to compensate for this effect is to modify the dispatching rule to look back in time some multiple of the setup time for each queue. When a machine becomes available, it next runs jobs from whichever queue was most behind back then. If the machine last processed jobs in queue (j,k) and the setup time required to process jobs from queue (j',k') is $s_{j,k,j',k'}$, then the machine should process jobs from queue (j',k') for which $\tilde{u}_{j,k}(t - \upsilon s_{j,k,j',k'}) - \hat{u}_{j,k}(t)$ is largest. Here υ is a

parameter that allows the schedule to avoid set-ups unless the current queue is significantly ahead of some other queue in keeping up with the optimal fluid solution. When $\upsilon = 0$, we have the naive approach. When υ is quite large and self-setups can be eliminated, so $s_{j,k,j',k'} > s_{j,k,j,k}$ for all $(j',k') \neq (j,k)$, we have the exhaustive service approach.

Figures 9.1 through 9.3 (laid side by side) show this simulation's event graph (using conventions defined by Schruben [1997] with minor departures by the author). Simulation state changes are represented by rectangular event vertices in this graph. Event vertex parameters, if any, are given in parentheses. Logical and dynamic relationships between pairs of events are represented in the graph by edges (arrows) between event vertices. Figure 9.1 shows the events that read the input data and set up the simulation. Figure 9.2 shows the events that govern the routing and processing of product units through the system. Figure 9.3 shows the events that allocate the machines to different buffers. Further description of the simulation can be found in Appendix B.

Table 9.1 is the input data file for our example network, Lines beginning with // are comments.

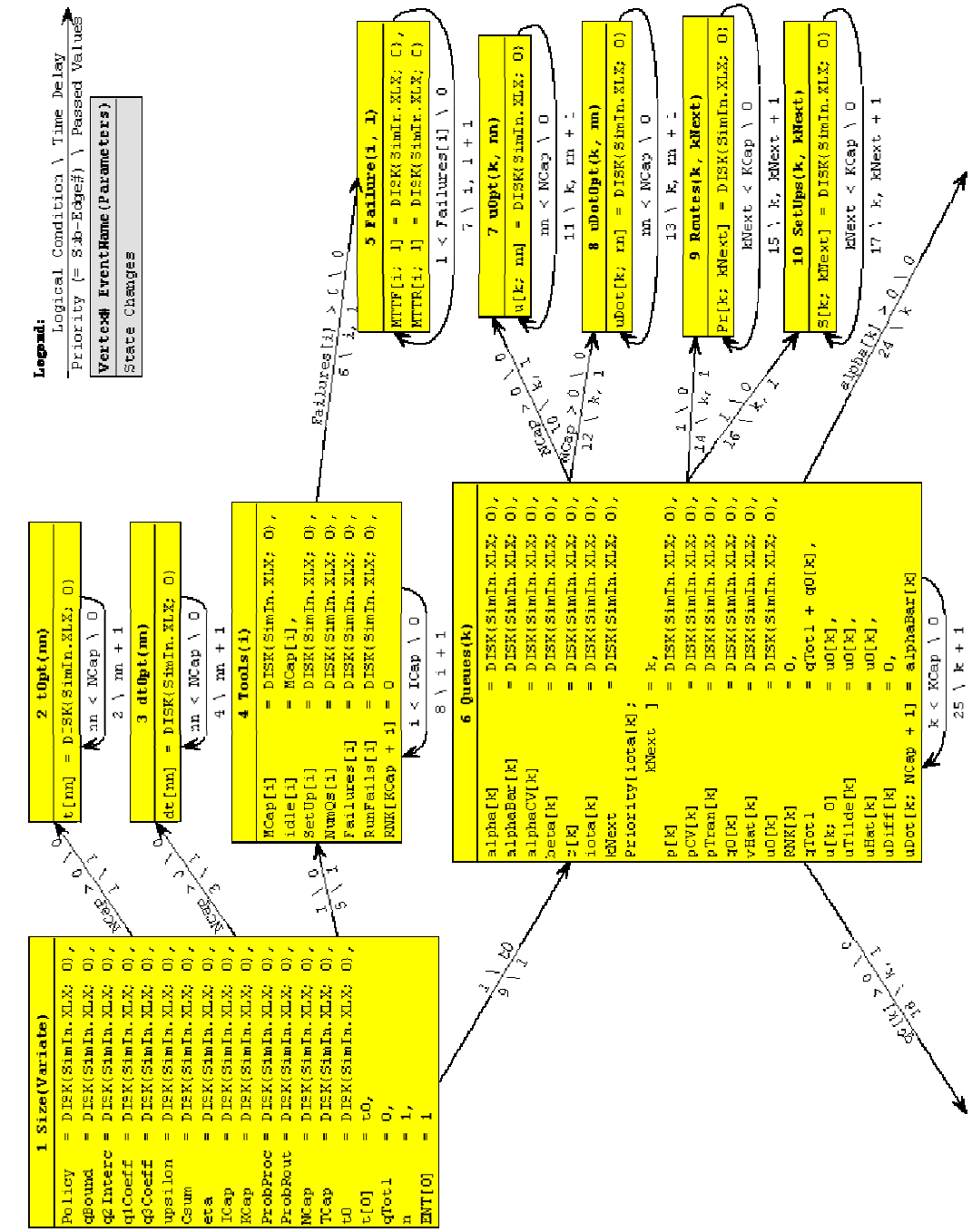

Figure 9.1: Simulation Event Graph: Input and Set-Up Events

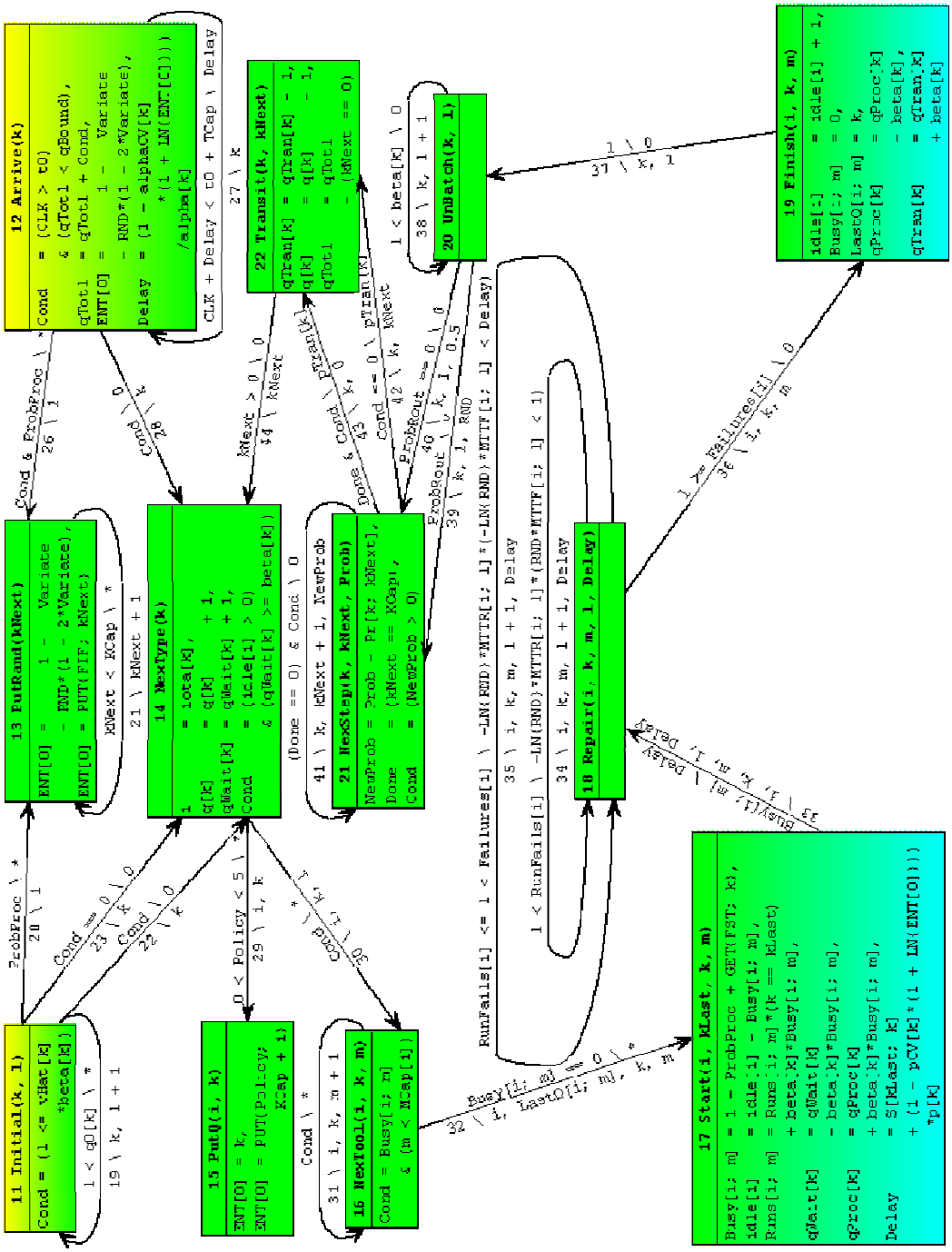

Figure 9.2: Simulation Event Graph: Unit Routing and Processing Events

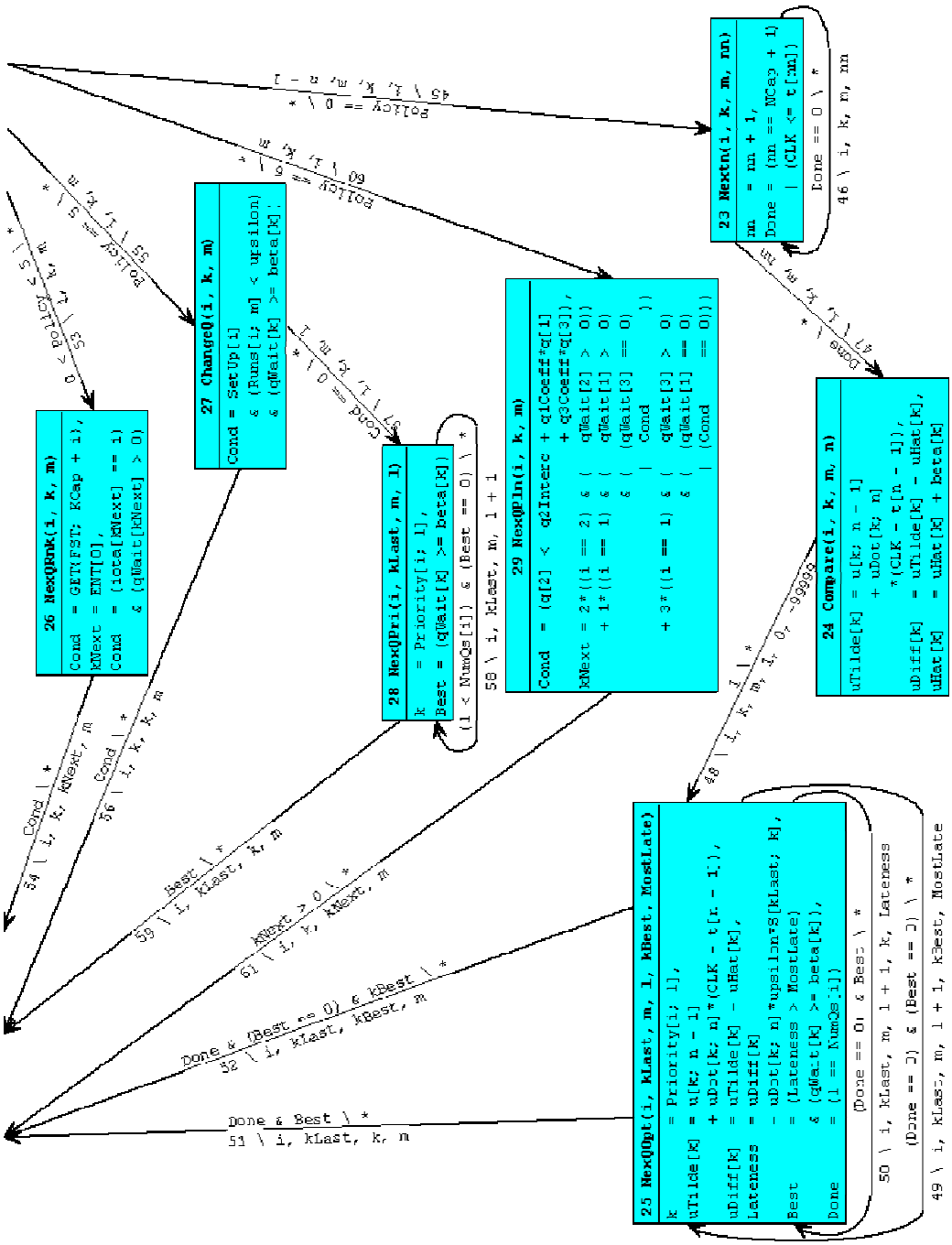

Figure 9.3: Simulation Event Graph: Machine Allocation Events

Table 9.1: Input Data File for Example Network

// Policy	qBound																					
5	999999																					
// q2Interc	q1Coeff	q3Coeff	upsilon	Csum	eta	lCap	KCap	ProbProc	ProbRout	NCap	TCap	t0	t[nn]		dt[nn]							
5	0	0	0	0	1	2	3	1	0	2	100000	0	11.9	128	11.9	116.1						
// MCap[i]	SetUp[i]	NumQs[i]	Failures[i]	RunFails[i]	MTTF[i; l]	MTTR[i; l]																
1	0	2	0	0																		
1	0	1	0	0																		
// alpha[k]	alphaBar[k]	alphaCV[k]	beta[k]	c[k]	iota[k]	Priority[i; j]	p[k]	pCV[k]	pTran[k]	q0[k]	vHat[k]	u0[k]	u[k; nn]		uDot[k; nn]		Pr[k; kNext]			S[k; kNext]		
1	1	1	1	1	1	1	0.2	1	0	1	0	17	17	146	0	1.1111111111	0	1	0	0	0	0
0	1	1	1	1	2	1	0.9	1	0	9	1	8	17	146	0.756302521	1.1111111111	0	0	1	0	0	0
0	1	1	1	1	1	2	0.7	1	0	8	1	0	17	146	1.428571429	1.1111111111	0	0	0	0	0	0

10. Computational Results

10.1 Fluid Model: Solution Methods

Each of the following figures show the performance of three different algorithms that solve the fluid formulations. Figure 10.1 shows the performance on our small example network with two machine types and three buffers. Figure 10.2 shows the performance on a network with five machine types and twenty buffers that was randomly generated by Weiss [2001]. Figure 10.3 shows the performance on a generic semiconductor process flow for 300-mm wafer fabrication with 281 machines among sixty types and 316 buffers that was developed by Quinn & Bass [1999] and extended by Campbell & Ammenheuser [1999] and by Stanley et al. [2001] at International SEMATECH.

In each figure, the horizontal axis shows the cumulative CPU time used by the GAMS solver (not including the GAMS compilation and execution time). The vertical axis shows (at each step) the deviation of the objective function value (from that of the optimal SCLP or QP solution) normalized so that the makespan solution (which all three algorithms use as a starting point) has a deviation of one.

- Shown in red diamonds is the performance of the finite quadratic programming (QP) algorithm (which always restarts at the optimal makespan solution as described in Chapter 6 and Section 7.3) that optimizes over the full horizon of the fluid formulation.
- Shown in blue squares is the performance of the linear programming (LP) algorithm (described in Section 7.1) that again optimizes over the full horizon of the fluid formulation.

- Shown in green triangles is the performance of the finite linear programming (LP) heuristic algorithm (described in Section 7.4) that only optimizes one time breakpoint at each step.

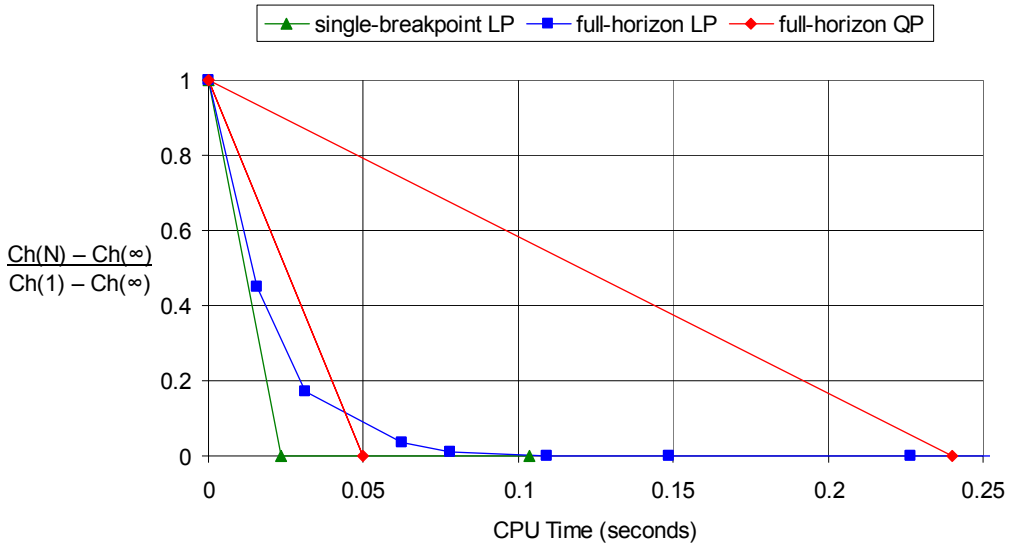

Figure 10.1: Algorithm Performance: Example Network

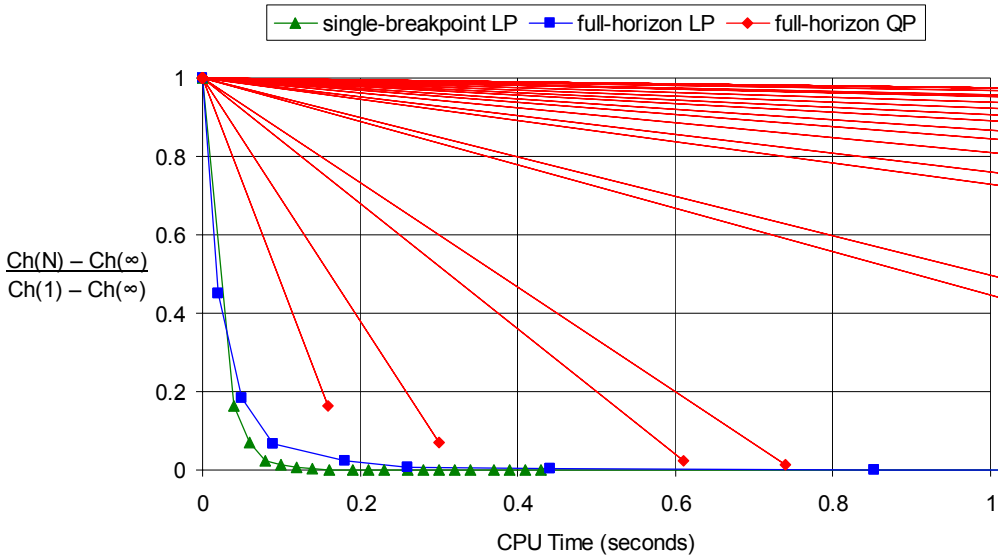

Figure 10.2: Algorithm Performance: 5-Machine 20-Buffer Network

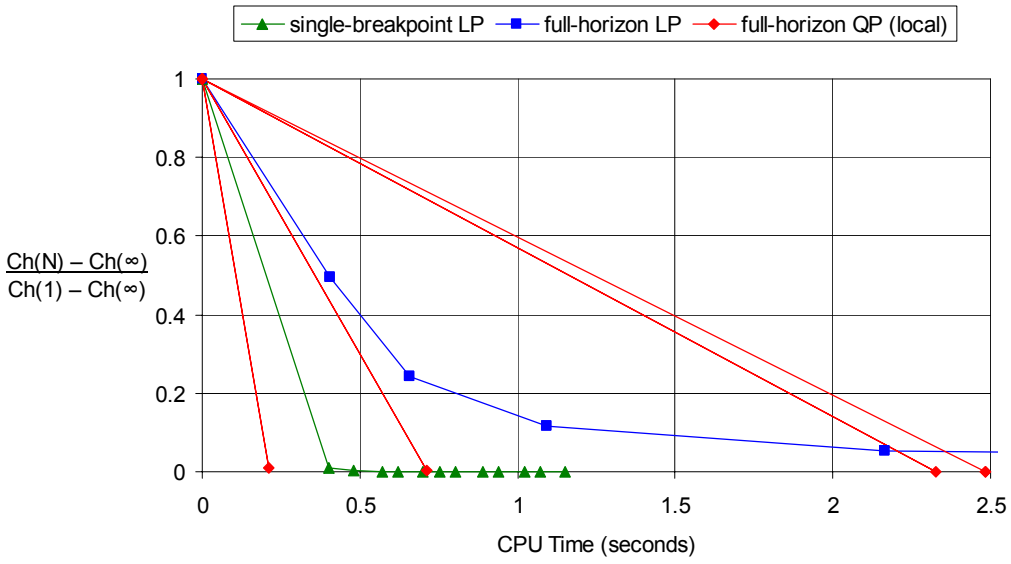

Figure 10.3: Algorithm Performance: 281-Machine 316-Buffer Wafer Fab

CONOPT was the solver for the LP formulations, and BARON was used for the QP formulation. However, BARON failed to find an improved solution on the 300-mm wafer fab data set, so CONOPT was used instead to find (what turned out to be quite good) local optima (which explains why the QP algorithm seemed to outperform the LP algorithms in Figure 10.3).

In general, the single-breakpoint LP heuristic seems superior to the other algorithms, consistently finding solutions with objective function values no more than 0.022% above the QP optimal solution (in the author's limited testing). On the other hand, a perfectly optimal solution is not needed for the fluid problem which is a relaxation (or abstraction) of the real deterministic (or stochastic) job-shop scheduling problem. Even including the GAMS compilation and execution time, the single-breakpoint LP heuristic produced a good solution for the huge 300-mm wafer fab data set in less than eight seconds (fast enough that it can be used for real-time dispatching in a wafer fab).

Combinations of algorithms were also investigated. Performing a QP solution after the single-breakpoint LP heuristic concluded or after each step of the single-breakpoint LP heuristic produced better results but not significantly so.

10.2 SIMULATION: SCHEDULING AND DISPATCHING POLICIES

Sigma can generate a version of the simulation in C, which can be compiled to run very fast. The experimental design used for this research involved (for each data set) twenty runs (ten pairs of random-number antithetic variates) with 100,000 arrivals in each run. Each set of twenty runs takes about five minutes on a Pentium III.

Figure 10.4 shows the performance (means and 95% confidence intervals) of several scheduling and dispatching policies on our small example network with two machine types and three buffers. The Threshold(l) policy is like that proposed by Harrison and Wein [1989] where the first machine serves the first buffer if and only if the last buffer is empty or the second buffer contains fewer than l items. Thus, a Threshold(0) policy is the same as a last-buffer-first-served (LBFS) policy. As $l \to \infty$, the Threshold(l) policy converges to a first-buffer-first-served (FBFS) policy. As predicted by the sequence of finite-dimensional linear programming (LP) formulations for the CTMDP (described at the end of Chapter 3), the lowest average WIP level (17.3 ± 0.3) was given by a Threshold(6) policy (although the mean was not significantly different for $5 \le l \le 9$).

Other policies that did only moderately well included first-in-first-out (FIFO), last-in-first-out (LIFO), and fluid-based real-time dispatching (not a fluid following method, but driven by an approximation of all fluid solutions based on the scale invariance ideas presented in Section 4.1). This might seem to suggest that fluid-based real-time dispatching methods are not very effective. However, the fluid model is a limit as the WIP in a stochastic model grows, so systems that are more heavily populated (not to be confused with having higher traffic intensities) than the example problem would be better approximated by fluid models (since their granularity is finer). Also, the type of CTMDP model that suggested the Threshold(l) policy is not practical for systems more complicated than the example problem. In the extreme example of a semiconductor wafer fab

(with hundreds of machine types and buffers and total WIP in the range of tens or hundreds of thousands), fluid models might function quite well. Fluid models may be even more effective in environments that combine continuous and discrete processing (such as pharmaceutical manufacturing).

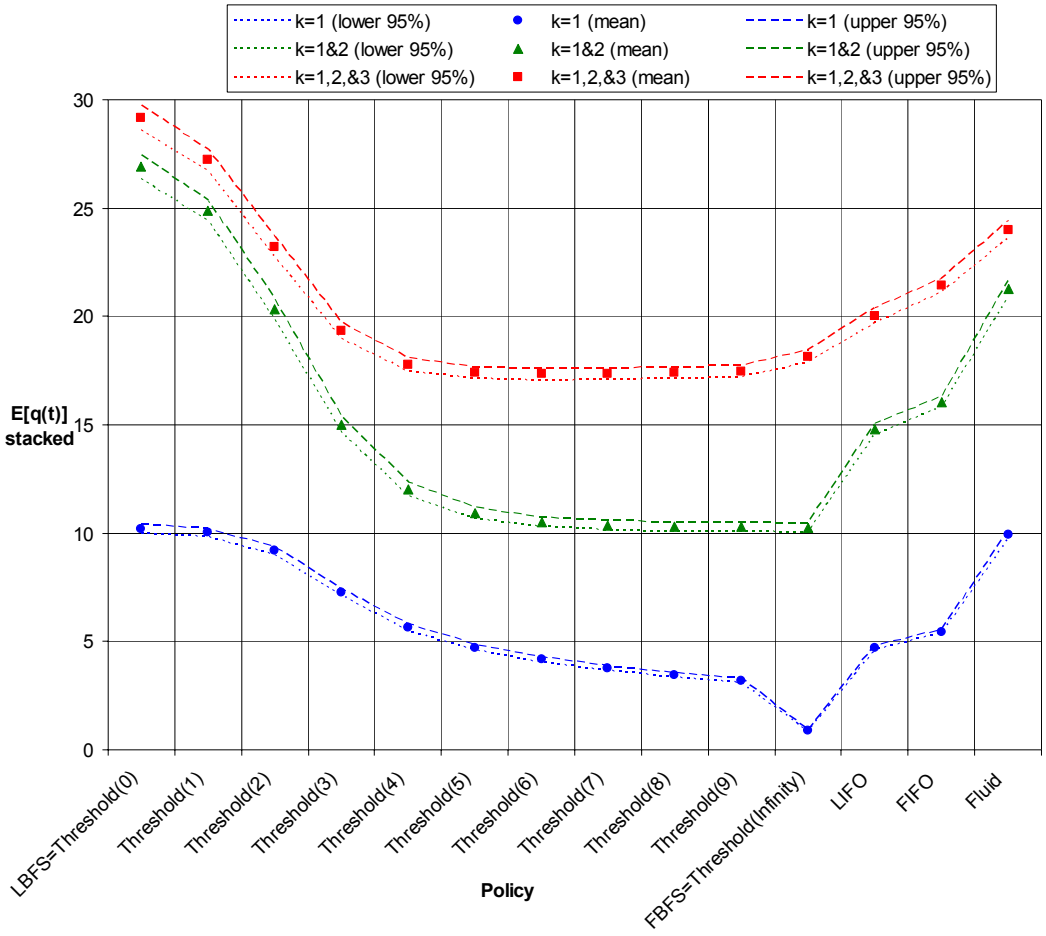

Figure 10.4: Scheduling/Dispatching Policy Performance: Example Network

11. Conclusion

11.1 New Results

As far as the author knows, the following ideas first came to light as a consequence of the research described in this document:

- A fluid queueing model is a relaxation of the deterministic job shop scheduling problem. Others have shown that a fluid queueing model is a limit of a stochastic queueing model and that a fluid queueing model can give a lower bound on makespan when there are no ongoing arrivals. However, no one has shown that the general deterministic job shop scheduling problem (presented in Chapter 2) with any of the three objective functions has a fluid queueing model relaxation. The fact that the fluid queueing model is a relaxation of the deterministic job shop scheduling problem is not as significant as it might sound. Although the fluid queueing model with the makespan objective function does give a useful lower bound, the lower bound given by the other two objective functions is arbitrarily small for a large enough planning horizon.
- A small linear programming (LP) algorithm (described in Section 7.4) that only optimizes one time breakpoint at each step quickly provides a good heuristic solution.
- Starting assignments can be set with a mixed-integer program based on the results of the fluid queueing model.

- Practical techniques exist for adapting optimal solutions of fluid models to discrete queueing networks (including methods for comprehending batching and for avoiding sequence-dependent set-ups).

11.2 FUTURE WORK

For this work to be useful in practice in wafer fabs and other factories, we need to know how often to recalculate a fluid solution. Possible candidates include:

- Daily or shiftly (with setups minimized over the ensuing period).
- Each time a machine breaks down or is repaired (or at other shocks to the system).
- Each time a machine finishes processing a lot and has a choice of buffers from which to draw.
- Some other time interval determined by simulation.

To resolve this, real-time fluid-based dispatching must be implemented in software so that a simulation (or a factory control system) can recalculate a fluid solution at any time. Also, due date constraints must be included, with lateness allowed in the constraints and/or penalized in the objective function.

Two issues (of more academic interest) also ought to be resolved. The first is to determine under which conditions a QP optimal solution drains after time T^*. The second is to prove or disprove that the optimal holding cost in the QP formulation is strictly decreasing in the number of time intervals until enough are used, then is constant for any larger number of time intervals. We have assumed this in our finite quadratic programming solution method.

Other research opportunities that could be pursued include the following directions:

- Duplicate the theory in Chapters 5 through 8 using the maximum immediate workload objective function.
- Determine how to set all the other parameters (besides how often to recalculate a fluid solution) that are needed to implement fluid-based dispatching (including parameters for setup avoidance, partial batching etc.).
- Verify which of the math programming methods in Chapters 5 through 8 works best for a wide variety of problem sizes and configurations.
- See if integrating other algorithms (such as those by Weiss [2002] and Fleischer and Sethuraman [2003]) with the methods presented here provides faster solutions.
- Attempt to find good lower bounds on the optimality gap in the math programming algorithms.

11.3 SUMMARY

Fluid models are potentially useful methods for finding good schedules for complex manufacturing systems and are more substantively grounded in queueing theory than other heuristics. Large-scale fluid model problems can be solved quickly, and translation issues are tricky but not insurmountable.

Appendix A: GAMS Code

A.1 EXAMPLE NETWORK DATA FILE

```
SCALARS
    I               number of machine types
    /2/
    K               number of job stages
    /3/
    N               number of time intervals
    /5/
    t0              initial time break-point
    /0/;

PARAMETERS
    m(Machines)     number of machines of each type
    /i1 1
     i2 1/
    a(Machines)     average availability of each machine type
    /i1 1
     i2 1/
    iota(Queues)    type of machine assigned to each job stage
    /k1 1
     k2 2
     k3 1/
    u0(Queues)      total number of units processed before t0
    /k1 0
     k2 0
     k3 0/
    p(Queues)       machine processing times
    /k1 0.2
     k2 0.9
     k3 0.7/
    c(Queues)       holding costs
    /k1 1
     k2 1
     k3 1/
    alpha(Queues)   external arrival rates
    /k1 1
     k2 0
     k3 0/
    q0(Queues)      initial queue lengths
    /k1 1
     k2 9
     k3 8/;

TABLE Pr(Queues, Stages) probabilistic routing matrix
```

```
       k1 k2 k3
    k1  0  1  0
    k2  0  0  1
    k3  0  0  0;
```

A.2 MAIN GAMS PROGRAM

```
$TITLE Linear and/or Quadratic Programs for Fluid Relaxation
$OFFUPPER
$INLINECOM { }
$OFFLISTING
$OFFSYMXREF
$OFFSYMLIST

* By Ron Billings
* Soli Deo Gloria
* Copyright 2002, 2003 FABQ.com

*****************************************************************

* Declarations and Definitions for Algorithm Parameters:

SCALARS
    OutputLevl level of output in listing (-1 or 0 or 1 or 2)
    /1/
    Exponterp  exponent for interpolating makespan solution
    /2.38780419/
    OpTolernce tolerance for deciding when optimality is reached
    /0.999999999/
    GlobalConv convergence criteria for global NLP optimization
    /0.01/
    eta        minimize holding cost (1) or maximum workload (0)
    /1/
    NormMIP    one (1) or infinity (0) norm at startup else (-1)
    /-1/
    Idling     number of idle tools allowed at startup (0 or INF)
    /0/
    VarBound   bound variables (1) or not (0)
    /1/
    dtBound    upper bound (times Tstar) on time interval length
    /4/
    NBound     upper bound (times given N) on number of intervals
    /100/
    NStart     start at N = 1 (1) or N given in data set (0)
    /1/
    Method     algorithm for subdividing time intervals
    /1/;
*               0 = Interpolate makespan solution for one N value
*                   and do full-horizon QP.
```

```
*              1 = For each improving N: Interpolate makespan
*                  solution, and do full-horizon QP.
*              2 = For each improving N:
*                  Interpolate N - 1 solution in each interval,
*                  do single-time-interval LP, and keep the Best.
*              3 = Do #2, then full-horizon QP on final solution.
*              4 = Do #2, but also do full-horizon QP for each N.
*              5 = For each improving N: Interpolate N/2 solution
*                  in each interval, and do full-horizon LP.

*eta = EPS;
*Exponterp = 1.380014616;

*****************************************************************

* Initial Options:

*OPTION  MIP       = OSL;
OPTION MIP        = CPLEX;
*OPTION MIP        = XPRESS;

OPTION   LP      = CONOPT;
*OPTION LP        = CPLEX;
*OPTION LP        = BDMLP;
*OPTION LP        = MINOS;
*OPTION LP        = OSL;
*OPTION LP        = SNOPT;

*OPTION  NLP      = BARON;
OPTION  NLP      = CONOPT;
*OPTION  NLP      = MINOS;
*OPTION  NLP      = PATHNLP;
*OPTION  NLP      = SNOPT;
*OPTION  NLP      = OQNLP;

OPTIONS ITERLIM  = 10000,
        RESLIM   = 10000,
        LIMROW   = 0,
        LIMCOL   = 0,
        SYSOUT   = OFF,
        SOLPRINT = OFF,
        DECIMALS = 8
        PROFILE  = 0;

FILE FluidOut in Excel format / FluidOut.xlx /;
FluidOut.AP =     1;
FluidOut.PC =     6;
FluidOut.PW = 32767;
FluidOut.ND =    10;
```

```
FluidOut.NR =     0;
FluidOut.NZ =     0;
FluidOut.NW =    12;
PUT FluidOut;
IF (OutputLevl >= 1,
    PUT SYSTEM.DATE SYSTEM.TIME eta Method;
    );

*****************************************************************

* Set Declarations:

SETS
    Machines          possible machine type indices
    /i1*i999/
    Machine(Machines) actual machine type indices
    Queues            possible job stage indices
    /k1*k999/
    Queue(Queues)     actual job stage indices
    Times             possible break-point or interval indices
    /n0*n999/
    TimePnt(Times)    actual time break-point indices
    TimeInt(Times)    actual time interval indices;

ALIAS(Times , Time  );
ALIAS(Stages, Queues);
ALIAS(Stage , Queue );

*****************************************************************

* Declarations for Full-Horizon Data to be Computed Immediately:

SCALARS
    cq0       c'*q0    initial weighted holding cost
    wmax0     initial maximum workload
    ChCmax    initial makespan solution holding cost objective
    CwCmax    initial makespan solution workload objective
    CsumCmax  weighted sum of ChCmax and CwCmax
    Tstar     minimal makespan
    pMin      smallest machine processing time of all stages
    etah      weight for Ch in objective
    etaw      weight for Cw in objective;

PARAMETERS
    IPT(Queues, Stages)       I - P'
    cBar(Queues)              (I - P )*c        keeping costs
    alphaBar(Queues)          upstream arrival rates
    B(Machines, Queues)       assignments of machines to stages
    Bq0(Machines)             B*q0
```

```
    MinmBq0(Machines)          minimum of m and Bq0
    am(Machines)               D(a)*m
    Bp(Machines, Queues)       B*D(p)
    BpalphaBar(Machines)       Bp*alphaBar
    chi(Machines)              am - BpalphaBar
    uFinal(Queues)             qBar0 + u0 + dtBound*alphaBar
    vFinal(Queues)             D(p)*alphaBar
    yDotFinal(Machines)        am - B*vFinal
    w0(Machines)               Inv[D(am)]*Bp*q0  initial workload
    BpqBar0(Machines)          Bp*qBar0
    wBar0(Machines)            Inv[D(am)]*BpqBar0 initial upstream
    Tmax(Machines)             Inv[D(chi)]*BpqBar0
    amBpIPT(Machines, Queues)  Inv[D(am)]*Bp*IPT
    rho(Machines)              stability condition metrics
    OutBound(Queues)           bound on rate units transfer out
    InBound(Queues)            bound on rate units transfer in
    qBarBound(Queues)          bound on qBar
    wMaxBound                  bound on wMax;

**************************************************************

* Input Data Declarations and Definitions:

*$INCLUDE "c:\GAMSCode\Florin.gms"
*$INCLUDE "c:\GAMSCode\Florand.gms"
*$INCLUDE "c:\GAMSCode\Hasen.gms"
*$INCLUDE "c:\GAMSCode\Hasen40.gms"
*$INCLUDE "c:\GAMSCode\Hasenrand.gms"
*$INCLUDE "c:\GAMSCode\MiniFab.gms"
*$INCLUDE "c:\GAMSCode\reentrnt.gms"
$INCLUDE "c:\GAMSCode\300mm.gms"

**************************************************************

* Dynamic Set Definitions:

Machine(Machines) = YES$(ORD(Machines) <= I);
Queue  (Queues  ) = YES$(ORD(Queues  ) <= K);

**************************************************************

* Linear-Equation Solver Model Declarations:

FREE VARIABLES
    VarLinEqns    dummy objective variable
    aBar(Queues)  upstream arrival rates
    qBar0(Queues) initial upstream queue lengths
    u0Min(Queues) minimal initial number of units processed
    InBar(Queues) bound on rate units transfer in upstream;
```

```
EQUATIONS
    ObjLinEqns    dummy objective function
    alphaBarEq    compute alphaBar
    qBar0Eq       compute qBar0
    u0MinEq       compute u0Min
    InBarEq       compute InBar;

*****************************************************************

* Linear-Equation Solver Model Definitions:

ObjLinEqns  ..
    VarLinEqns =E=   SUM(Queue,                    aBar(Queue)
                                                 + qBar0(Queue)
                                                 + u0Min(Queue));

alphaBarEq(Queue) .. SUM(Stage, IPT(Queue, Stage)*aBar( Stage))
    =E=                                          alpha(Queue);

qBar0Eq(   Queue) .. SUM(Stage, IPT(Queue, Stage)*qBar0(Stage))
    =E=                                          q0(   Queue);

u0MinEq(   Queue) .. SUM(Stage, IPT(Stage, Queue)*u0Min(Stage))
    =E=              SUM(Stage,  Pr(Queue, Stage)*q0(   Stage));

InBarEq(   Queue) .. SUM(Stage, IPT(Queue, Stage)*InBar(Stage))
    =E=                                        InBound(Queue);

MODEL LinearEqns /ObjLinEqns, alphaBarEq, qBar0Eq, u0MinEq,
                                                   InBarEq/;

*****************************************************************

* Definitions for Full-Horizon Data to be Computed Immediately:

IPT(        Queue  , Queue) = 1;
IPT(        Queue  , Stage) =            IPT(Queue, Stage)
                                       - Pr(Stage, Queue);
cBar(               Queue) = SUM(Stage, IPT(Stage, Queue)
                                       *c(  Stage       ));
cq0                        = SUM(Queue, c( Queue)
                                       *q0(Queue));
B(         Machines, Queues)$(ORD(Machines) = iota(Queues))
                           = 1;
Bq0(       Machine        ) = SUM(Queue, B(Machine, Queue)
                                       *q0(        Queue));
MinmBq0(   Machine        ) = MIN(  m(Machine),
                                 Bq0(Machine));
```

```
am(        Machine        ) = m(Machine)
                              *a(Machine);
Bp(        Machine , Queue) = B(Machine, Queue)
                              *p(         Queue);
OutBound(            Queue) = SUM(Machine, Bp(Machine, Queue)
                                           *am(Machine)       );
InBound(             Queue) = SUM(Stage, Pr(Stage, Queue)
                                          *OutBound(Stage) );
SOLVE LinearEqns USING LP MINIMIZING VarLinEqns;
alphaBar(Queue) = aBar.L(Queue);
IF (SUM(Queue, u0(Queue)) = 0,
    u0(Queue) = u0Min.L(Queue);
    );
BpalphaBar(Machine        ) = SUM(Queue, Bp(Machine, Queue)
                                          *alphaBar( Queue));
chi(       Machine        ) = am(        Machine)
                              - BpalphaBar(Machine);
vFinal(              Queue) = p(Queue)*alphaBar(Queue);
yDotFinal( Machine        ) = am(          Machine)
                              - SUM(Queue, B(Machine, Queue)
                                          *vFinal(    Queue));
w0(        Machine        ) = SUM(Queue, Bp(Machine, Queue)
                                          *q0(         Queue))
                               /am(Machine);
BpqBar0(   Machine        ) = SUM(Queue, Bp(Machine, Queue)
                                          *qBar0.L(    Queue));
wBar0(     Machine        ) = BpqBar0(Machine)
                               /am(     Machine);
amBpIPT(   Machine , Queue) = SUM(Stage, Bp(Machine, Stage)
                                          *IPT(Stage,  Queue))
                               /am(Machine);
Tmax(      Machine        ) = BpqBar0(Machine)
                               /chi(    Machine);
Tstar                        = MAX(0,
                                   SMAX(Queue  , -qBar0.L(Queue )
                                                 /alphaBar(Queue )),
                                   SMAX(Machine, Tmax(   Machine)));
wmax0                        =   SMAX(Machine, w0(     Machine));
ChCmax                       = cq0  *Tstar/2;
CwCmax                       = wmax0*Tstar/2;
CsumCmax                     = 1;
etah                         = .5*     eta /ChCmax;
etaw                         = .5*(1 - eta)/CwCmax;
rho(       Machine        ) = BpalphaBar(Machine)
                               /am(        Machine);
NBound                       = N*NBound;
dtBound                      = MIN(3E+7, Tstar*dtBound);
uFinal(Queue)                = qBar0.L(Queue) + u0(Queue)
                             + Tstar*dtBound*alphaBar(Queue);
```

```
qBarBound(Queue)            = qBar0.L(Queue)
                            + Tstar*dtBound*(  alphaBar(Queue)
                                             + InBar.L( Queue));
qBarBound(Queue)            = MIN(3E+7, qBarBound(Queue));
wMaxBound                   = SMAX(Machine, SUM(Queue,
                               amBpIPT(Machine, Queue)
                               *qBarBound(      Queue)));
wMaxBound                   = MIN(3E+7, wMaxBound);
*****************************************************************

* Declarations for Full-Horizon Data to be Computed Later:

SCALAR
    Ch                      total weighted holding cost objective
    Cw                      total maximum workload objective
    Cmax                    makespan of current solution
    deltat                  fixed time-period length in LP
    SolveTime               time for solver computations;

PARAMETERS
    t(               Times) time break-points
    cq(              Times) c'*q
    BpqBar(Machines, Times) Bp*qBar
    yDot(  Machines, Times) number of machines idle in intervals
    w(     Machines, Times) workloads at time break-points
    u(       Queues, Times) total processed by time break-points
    uDot(    Queues, Times) processing rate during time intervals
    v(       Queues, Times) # of machines working each stage;

*****************************************************************

* Full-Horizon Model Declarations:

POSITIVE VARIABLE
    dt(          Times) time-period length;

FREE VARIABLES
    Csum                weighted sum of Ch and Cw
    qBar(Queues, Times) upstream queue lengths at break-points
    wmax(        Times) maximum workloads at time break-points;

EQUATIONS
    ObjQP  compute Csum with quadratic terms in separate file
    ObjNLP compute Csum with quadratic terms in objective fcn
    ObjLP  compute Csum with linear terms in objective function
    du     units processed during time period
    dy     accumilated machine idleness during time period
    q      queue lengths at time break-points
    wDev   workload deviations at time break-points;
```

```
*****************************************************************

* Full-Horizon Model Definitions:

ObjNLP  ..
    Csum =E= SUM(Times$TimeInt(Times),
             dt(Times)*(  etah*SUM(    Queue,
                                cBar(   Queue)
                               *(  qBar(Queue, Times - 1)
                                 + qBar(Queue, Times    )))
                         + etaw*(  wmax(       Times - 1)
                                 + wmax(       Times    ))));

ObjLP  ..
    Csum =E= SUM(Times$TimeInt(Times),
                deltat*(  etah*SUM(    Queue,
                                cBar(   Queue)
                               *(  qBar(Queue, Times - 1)
                                 + qBar(Queue, Times    )))
                         + etaw*(  wmax(       Times - 1)
                                 + wmax(       Times    ))));

du(Queue  , Times)$TimeInt(Times) ..
        alphaBar(Queue)*dt(Times)  + qBar(Queue, Times - 1)
                                   - qBar(Queue, Times    )
    =G= 0;

dy(Machine, Times)$TimeInt(Times) ..
           chi(Machine)*dt(Times)
         - SUM(Queue, Bp(Machine, Queue)
                     *(             qBar(Queue, Times - 1)
                                   - qBar(Queue, Times    )))
    =G= 0;

q(Queue  , TimePnt) ..
    SUM(Stage,   IPT(Queue , Stage)* qBar(Stage, TimePnt  ))
    =G= 0;

wDev(Machine, TimePnt) ..
                                                 wmax(TimePnt  )
    - SUM(Queue, amBpIPT(Machine, Queue)*qBar(Queue, TimePnt  ))
    =G= 0;

MODEL HorizonNLP /ObjNLP, du, dy, q, wDev/;

MODEL HorizonLP  /ObjLP,  du, dy, q, wDev/;

$ONTEXT
```

```
* Kludgy GAMS QPWRAP linear objective function

ObjQP  ..
    Ch =E= EPS*SUM(TimeInt,  dt(TimeInt)
                           + SUM(Queue, qBar(Queue, TimeInt)));

* Create separate file for quadratic terms in objective function

FILE qp /qmatrix.txt/;
qp.pc =  5;
qp.nr =  2;
qp.nd = 10;
qp.nw =  0;
PUT qp 'Quadratic terms for objective function';
LOOP((Time, Queue)       $(    (cBar(Queue) <> 0            )
                           AND TimeInt(Time)
                           AND (ORD(Time)  <= N           )),
    PUT / 'dt'         Time.tl
          'qBar' Queue.tl Time.tl
          (cBar(Queue)/2);
    );
LOOP((Time, Times, Queue)$(    (cBar(Queue) <> 0            )
                           AND TimeInt(Time)
                           AND (ORD(Time)   = ORD(Times) + 1)),
    PUT / 'dt'         Time.tl
          'qBar' Queue.tl Times.tl
          (cBar(Queue)/2);
    );
PUTCLOSE qp;

MODEL HorizonQP  /ObjQP, du, dy, q/;

$OFFTEXT

*****************************************************************

* Declarations of One-Time-Break-Point Data to be Computed Later:

SCALARS
    constLP            constant term in LP objective
    dtCoef             coefficient of dtInterval in LP objective
    wmaxCoef           coefficient of wmaxIntrvl in LP objective
    nNow               current time interval number
    nBest              best time interval number from LPs
    dtBest             best time interval length from LPs
    wmaxBest           best maximum workload from LPs
    CBest              best objective from LPs
    CLast              previous best objective
```

```
    Need4MoreT         change to Csum if time interval increases;

PARAMETERS
    qBarBest(Queues)   best qBar from LPs
    qBarCoef(Queues)   coefficient of qBarIntrvl in LP objective;

*****************************************************************

* One-Time-Break-Point Model Declarations:

FREE VARIABLES
    CsumIntrvl         total weighted holding cost
    duValue(Queues)    units processed during time period
    dyValue(Machines)  machine idleness during time period
    qBarIntrvl(Queues) upstream queue lengths at break-point
    wmaxIntrvl         maximum workload at break-point;

POSITIVE VARIABLE
    dtInterval         time period length before break-point;

EQUATIONS
    ObjIntrvl          compute CsumIntrvl
    duInterval         units processed during time period
    dyInterval         machine idleness during time period
    qInterval          queue lengths at time break-point
    wDevIntrvl         workload deviations at time break-point;

*****************************************************************

* One-Time-Break-Point Model Definitions:

ObjIntrvl  ..
    CsumIntrvl =E= constLP
                 + dtCoef*dtInterval
                 + SUM(Queue, qBarCoef(  Queue)
                             *qBarIntrvl(Queue))
                 + wmaxCoef*wmaxIntrvl;

duInterval(Queue) ..
    -alphaBar(Queue)*dtInterval
    + qBarIntrvl(Queue) =E= duValue(Queue);

dyInterval(Machine) ..
    chi(Machine)*dtInterval
    + SUM(Queue, Bp(Machine, Queue)
                *qBarIntrvl( Queue)) =E= dyValue(Machine);

qInterval(Queue) ..
    SUM(Stage, IPT(Queue , Stage)
```

```
              *qBarIntrvl( Stage)) =G= 0;

wDevIntrvl(Machine) ..
                                              wmaxIntrvl
    - SUM(Queue, amBpIPT(Machine, Queue)*qBarIntrvl(Queue))
    =G= 0;

MODEL IntervalLP
    /ObjIntrvl, duInterval, dyInterval, qInterval, wDevIntrvl/;

*****************************************************************

* Start-Up Model Declarations:

FREE VARIABLE
    NormvDev          norm of vDev;

POSITIVE VARIABLES
    yDotHat(Machines) number of machines idle at start-up
    vDev1(Queues)     absolute value of v minus vHat
    vDevInf(Machines) max over each machine type of vDev1;

INTEGER VARIABLE
    vHat(Queues)      discrete start-up machine assignments;

EQUATIONS
    ObjMIP            compute NormvDev
    PosDif            vHat minus v
    NegDif            v minus vHat
    BvHat             non-idling condition;

*****************************************************************

* Start-Up Model Definitions:

ObjMIP  ..
    NormvDev =E=      NormMIP *SUM(Queue  , vDev1(  Queue  ))
                + (1 - NormMIP)*SUM(Machine, vDevInf(Machine));

PosDif(Queue) ..
                 NormMIP *             vDev1(           Queue)
          + (1 - NormMIP)*SUM(Machine, B(      Machine, Queue)
                                      *vDevInf(Machine))
    =G=  vHat(Queue)
       - v(Queue, 'n1')/SUM(Machine, B(      Machine, Queue)
                                      *a(      Machine));

NegDif(Queue) ..
                 NormMIP *             vDev1(           Queue)
```

```
        + (1 - NormMIP)*SUM(Machine, B(      Machine, Queue)
                                    *vDevInf(Machine))
    =G= -vHat(Queue)
       + v(Queue, 'n1')/SUM(Machine, B(      Machine, Queue)
                                    *a(      Machine));

BvHat(Machine) ..
    SUM(Queue, B(Machine, Queue)
              *vHat(      Queue))
    + yDotHat(Machine) =E= MinmBq0(Machine);

MODEL StartUpMIP /ObjMIP, PosDif, NegDif, BvHat/;

*****************************************************************

* Run Commands:

IF (OutputLevl >= 2,
    OPTIONS SYSOUT   = ON,
            SOLPRINT = ON,
            PROFILE  = 1;
    HorizonNLP.SOLPRINT = 2;
    IntervalLP.SOLPRINT = 2;
    DISPLAY OutputLevl, Exponterp, OpTolernce, GlobalConv,
            NormMIP, eta, Idling, VarBound, dtBound, NBound,
            NStart, Method, I, K, N, m, a, iota, c, p, Pr, q0,
            alpha, IPT, alphaBar, qBar0.L, InBar.L, u0Min.L, u0,
            w0, wBar0, Tmax, wmax0, cBar, cq0, B, Bq0, MinmBq0,
            am, Bp, BpalphaBar, chi, amBpIPT, BpqBar0, Tstar,
            ChCmax, CwCmax, CsumCmax, etah, etaw, rho,
            OutBound, InBound, qBarBound, wMaxBound;
    );
HorizonNLP.OPTCR     = GlobalConv;
HorizonNLP.WORKSPACE = 999;
*HorizonNLP.OPTFILE   = 1;
*HorizonNLP.PRIOROPT  = 1;
*dt.PRIOR(Times)      = 2;
*IF (Method <= 1,
*    option solveopt = replace;
*    );
pMin = SMIN(Queue, p(Queue));
IF (VarBound = 1,
*    dt.LO('n1') = pMin;
    );
qBar.FX(Queue , 'n0') = qBar0.L(Queue);
q.L(    Queue , 'n0') = q0(Queue);
u(      Queue , 'n0') = u0(Queue);
vHat.L( Queue)        = 0;
wDev.L(Machine, 'n0') = w0(Machine) - wmax0;
```

```
wmax.FX(        'n0') = wmax0;
t(              'n0') = t0;
Cmax                  = Tstar;
deltat                = Tstar;
nBest                 = 0;
dtBest                = 0;
wmaxBest              = 0;
qBarBest(Queue)       = 0;
CBest                 = 2*CsumCmax;
CLast                 = INF;
TimeInt(Times)        = NO;
*t(TimeInt)            = 0;
SolveTime             = 0;
IF (NStart = 1,
    N = 1;
    );

*FOR (Exponterp = 1 TO 13 BY .125,
*WHILE ((N <= NBound),
WHILE ((CBest < CLast*OpTolernce) AND (N <= NBound),

    IF ((N = 1) OR (NStart = 0) OR (Method <= 1),

        NStart = 1;

*       Interpolate makespan solution for a feasible solution.
$ONTEXT
        Exponterp = N/2;
        IF (N - 1 > 1,
            Exponterp = SUM(Times$(    TimeInt(Times)
                                   AND (ORD(Times) - 1 < N - 1)),
                               LOG( t(Times)
                                   /Cmax)
                              *LOG( (ORD(Times) - 1)
                                   /(N          - 1)))
                         /SUM(Times$(    TimeInt(Times)
                                     AND (ORD(Times) - 1 < N - 1)),
                               POWER(LOG( (ORD(Times) - 1)
                                         /(N          - 1)), 2));
            );
$OFFTEXT
        IF (Method = 5,
            Exponterp = 1;
            deltat    = Tstar/N;
            );
        TimePnt(Times) = YES$(     ORD(Times) - 1 <= N );
        TimeInt(Times) = YES$(    (ORD(Times) - 1 <= N)
                              AND (ORD(Times) - 1 >  0));
        t(Times)$TimeInt(Times)
```

```
    = Tstar*((ORD(Times) - 1)/N)**Exponterp;
dt.L(Times)$TimeInt(Times) = t(Times) - t(Times - 1);
qBar.L(Queue, Times)$TimeInt(Times)
    = qBar0.L(Queue)
     *(1 - ((ORD(Times) - 1)/N)**Exponterp);
q.L(Queue, Times)$TimeInt(Times)
    = q0(Queue)
     *(1 - ((ORD(Times) - 1)/N)**Exponterp);
IF (VarBound = 1,
    dt.UP(         TimeInt) = dtBound;
    qBar.LO(Queue, TimeInt) = 0;
    qBar.UP(Queue, TimeInt) = qBarBound(Queue);
    wmax.LO(       TimeInt) = 0;
    wmax.UP(       TimeInt) = wMaxBound;
ELSE
    qBar.LO(Queue, TimeInt) = -INF;
    qBar.UP(Queue, TimeInt) =  INF;
    wmax.LO(       TimeInt) = -INF;
    wmax.UP(       TimeInt) =  INF;
    );
qBar.FX(Queue, Times)$(ORD(Times) - 1 = N) = 0;
wmax.FX(       Times)$(ORD(Times) - 1 = N) = 0;
w(Machine, Times)$TimeInt(Times)
    = w0(Machine)
     *(1 - ((ORD(Times) - 1)/N)**Exponterp);
wmax.L(Times)$TimeInt(Times)
    = wmax0
     *(1 - ((ORD(Times) - 1)/N)**Exponterp);
udot(  Queue  , TimeInt) = alphaBar( Queue)
                         + qBar0.L(  Queue)/Tstar;
du.L(  Queue  , TimeInt) = udot(  Queue  , TimeInt)
                          *dt.L(           TimeInt);
LOOP(Times$TimeInt(Times),
    u(Queue, Times)      = u(   Queue, Times - 1)
                         + du.L(Queue, Times    );
    );
v(     Queue  , TimeInt) = udot(  Queue  , TimeInt)
                           *p(     Queue);
ydot(  Machine, TimeInt) = chi(    Machine)
                         - BpqBar0(Machine)/Tstar;
dy.L(  Machine, TimeInt) = ydot(  Machine, TimeInt)
                          *dt.L(           TimeInt);
wDev.L(Machine, TimePnt) = wmax.L(    TimePnt)
                         - w(Machine, TimePnt);
Csum.L                   = CsumCmax;
Ch                       = ChCmax;
Cw                       = CwCmax;
CLast                    = CBest;
dt.M(          TimeInt) = 0;
```

```
        du.M(  Queue  , TimeInt) = 0;
        dy.M(  Machine, TimeInt) = 0;
         q.M(  Queue  , TimePnt) = 0;
        wDev.M(Machine, TimePnt) = 0;
        Csum.M                   = 0;

$INCLUDE "c:\GAMSCode\FluidOut.gms"

    ELSEIF Method <= 4,

*       Subdivide best time interval.
        TimePnt(Times)$(ORD(Times) - 1 = N) = YES;
        TimeInt(Times)$(ORD(Times) - 1 = N) = YES;
        IF (VarBound = 1,
            dt.UP(         TimeInt) = dtBound;
            qBar.LO(Queue, TimeInt) = 0;
            qBar.UP(Queue, TimeInt) = qBarBound(Queue);
            wmax.LO(       TimeInt) = 0;
            wmax.UP(       TimeInt) = wMaxBound;
        ELSE
            qBar.LO(Queue, TimeInt) = -INF;
            qBar.UP(Queue, TimeInt) =  INF;
            wmax.LO(       TimeInt) = -INF;
            wmax.UP(       TimeInt) =  INF;
            );
        qBar.FX(    Queue, Times  )$(ORD(Times) - 1 = N) = 0;
        wmax.FX(           Times  )$(ORD(Times) - 1 = N) = 0;
        FOR (nNow = (N - 1) DOWNTO (nBest + 1),
            qBar.L( Queue, Times  )$(ORD(Times) - 1 = nNow)
                = qBar.L(Queue, Times - 1);
            wmax.L(        Times  )$(ORD(Times) - 1 = nNow)
                = wmax.L(       Times - 1);
            );
        qBar.L(Queue, Times)$(ORD(Times) - 1 = nBest)
                = qBarBest(Queue);
        wmax.L(       Times)$(ORD(Times) - 1 = nBest)
                = wmaxBest;

        FOR (nNow = N DOWNTO (nBest + 2),
            dt.L(      Times)$(ORD(Times) - 1 = nNow)
                = dt.L(Times - 1);
            );
        dt.L(          Times)$(ORD(Times) - 1 = nBest + 1)
            = dt.L(    Times - 1) - dtBest;
        dt.L(          Times)$(ORD(Times) - 1 = nBest    )
            = dtBest;

        du.L(Queue, Times)$TimeInt(Times)
                           = alphaBar(Queue)*dt.L(Times    )
```

```
                        + qBar.L(  Queue,      Times - 1)
                        - qBar.L(  Queue,      Times    );
        uDot(Queue, TimeInt) = ( du.L(Queue  , TimeInt)
                              /dt.L(         TimeInt))
                             $(dt.L(         TimeInt) > 0);
        LOOP(Times$TimeInt(Times),
            u(Queue, Times)      = u(   Queue, Times - 1)
                                 + du.L(Queue, Times    );
            t(       Times)      = t(          Times - 1)
                                 + dt.L(       Times    );
            );
        v(   Queue, TimeInt) = uDot(  Queue  , TimeInt)
                              *p(     Queue)           ;
        w( Machine, TimePnt) = SUM(Queue, amBpIPT(Machine, Queue)
                                    *qBar.L(Queue, TimePnt));
        dy.L(Machine, Times)$TimeInt(Times)
                             = chi(Machine)*dt.L(Times)
                             - SUM(Queue, Bp(Machine, Queue)
                              *(  qBar.L(Queue, Times - 1)
                                - qBar.L(Queue, Times    )));
        yDot(Machine, TimeInt) = ( dy.L(Machine, TimeInt)
                                  /dt.L(         TimeInt))
                                 $(dt.L(         TimeInt) > 0);
        Ch = SUM(Times$TimeInt(Times),
            dt.L(Times)*SUM(      Queue,
                         cBar(    Queue)
                        *(  qBar.L(Queue, Times - 1)
                          + qBar.L(Queue, Times    ))))/2;
        Cw = SUM(Times$TimeInt(Times),
            dt.L(Times)*(  wmax.L(       Times - 1)
                         + wmax.L(       Times    )))/2;
        Csum.L = CBest;

$INCLUDE "c:\GAMSCode\FluidOut.gms"

    ELSEIF Method = 5,

        N = N/2;
*       Prune empty time intervals at end.
        wmaxBest = SUM(Times$(ORD(Times) - 1 = N - 1),
                       wmax.L(Times));
        WHILE (wmaxBest = 0,
            N = N - 1;
            wmaxBest = SUM(Times$(ORD(Times) - 1 = N - 1),
                           wmax.L(Times));
            );

*       Subdivide each time interval.
        FOR (nNow = N DOWNTO 1,
```

```
    qBarBest(       Queue)
        = SUM(Times$(ORD(Times) - 1 =   nNow),
          qBar.L(   Queue, Times));
    qBar.L(         Queue, Times)
                     $(ORD(Times) - 1 = 2*nNow)
        = qBarBest( Queue);
    qBarBest(       Queue)
        = (qBarBest(Queue)
           + SUM(Times$(ORD(Times) - 1 =   nNow - 1),
             qBar.L(Queue, Times)))/2;
    qBar.L(         Queue, Times)
                     $(ORD(Times) - 1 = 2*nNow - 1)
        = qBarBest( Queue);
    qBarBest(       Queue)
        = SUM(Times$(ORD(Times) - 1 =   nNow),
             u(   Queue, Times));
    u(              Queue, Times)
                     $(ORD(Times) - 1 = 2*nNow)
        = qBarBest( Queue);
    qBarBest(       Queue)
        = (qBarBest(Queue)
           + SUM(Times$(ORD(Times) - 1 =   nNow - 1),
             u(     Queue, Times)))/2;
    u(              Queue, Times)
                     $(ORD(Times) - 1 = 2*nNow - 1)
        = qBarBest( Queue);
    qBarBest(       Queue)
        = SUM(Times$(ORD(Times) - 1 =   nNow),
                v(Queue, Times));
    v(              Queue, Times)
                  $(   (ORD(Times) - 1 = 2*nNow    )
                    OR (ORD(Times) - 1 = 2*nNow - 1))
        = qBarBest( Queue);
    wmaxBest
        = SUM(Times$(ORD(Times) - 1 =   nNow),
               wmax.L(   Times));
    wmax.L(         Times)$(ORD(Times) - 1 = 2*nNow)
        = wmaxBest;
    wmaxBest
        = (wmaxBest
           + SUM(Times$(ORD(Times) - 1 =   nNow - 1),
                  wmax.L(Times)))/2;
    wmax.L(         Times)$(ORD(Times) - 1 = 2*nNow - 1)
        = wmaxBest;
    );
N = N*2;
deltat = deltat/2;
TimePnt(Times)$(ORD(Times) - 1 <= N) = YES;
TimeInt(Times)$(ORD(Times) - 1 <= N) = YES;
```

```
        TimeInt(Times)$(ORD(Times) - 1  = 0) = NO;
        dt.FX(TimeInt) = deltat;
        CLast = Csum.L;
        LOOP(Times$TimeInt(Times),
            t(       Times)      = t(          Times - 1)
                                 + dt.L(       Times    );
            );
        SolveTime = 0;

$INCLUDE "c:\GAMSCode\FluidOut.gms"

        );

    IF (N > 1,

        IF ((Method <= 1) OR (Method = 4),

*           Optimize this feasible solution with full-horizon QP.
            SOLVE HorizonNLP USING NLP MINIMIZING Csum;
            SolveTime = HorizonNLP.RESUSD;
            Cmax = SUM(TimeInt, dt.L(TimeInt));
            yDot(Machine, TimeInt) = ( dy.L(Machine, TimeInt)
                                     /dt.L(         TimeInt))
                                    $(dt.L(         TimeInt)>0);
            uDot(Queue  , TimeInt) = ( du.L(Queue  , TimeInt)
                                     /dt.L(         TimeInt))
                                    $(dt.L(         TimeInt)>0);
            LOOP(Times$TimeInt(Times),
                u(Queue, Times)      = u(   Queue, Times - 1)
                                     + du.L(Queue, Times    );
                t(       Times)      = t(          Times - 1)
                                     + dt.L(       Times    );
                );
            v(   Queue  , TimeInt) = uDot(  Queue  , TimeInt)
                                    *p(     Queue)           ;
            w(   Machine, TimePnt) = wmax.L(         TimePnt)
                                   - wDev.L(Machine, TimePnt);
            Ch = SUM(Times$TimeInt(Times),
                dt.L(Times)*SUM(      Queue,
                            cBar(     Queue)
                           *(  qBar.L(Queue, Times - 1)
                             + qBar.L(Queue, Times    ))))/2;
            Cw = SUM(Times$TimeInt(Times),
                dt.L(Times)*(  wmax.L(       Times - 1)
                             + wmax.L(       Times    )))/2;

$INCLUDE "c:\GAMSCode\FluidOut.gms"

        ELSEIF Method = 5,
```

```
*           Optimize this feasible solution with full-horizon LP.
            wmaxBest = SUM(Times$(ORD(Times) - 1 = N - 1),
                             wmax.L(Times));
            WHILE (wmaxBest > 0,
                N = N + 1;
                TimePnt(Times)$(ORD(Times) - 1 <= N) = YES;
                TimeInt(Times)$(ORD(Times) - 1 <= N) = YES;
                TimeInt(Times)$(ORD(Times) - 1  = 0) = NO;
                dt.FX(             TimeInt) = deltat;
                IF (VarBound = 1,
                    qBar.LO(Queue, TimeInt) = 0;
                    qBar.UP(Queue, TimeInt) = qBarBound(Queue);
                    wmax.LO(       TimeInt) = 0;
                    wmax.UP(       TimeInt) = wMaxBound;
                ELSE
                    qBar.LO(Queue, TimeInt) = -INF;
                    qBar.UP(Queue, TimeInt) =  INF;
                    wmax.LO(       TimeInt) = -INF;
                    wmax.UP(       TimeInt) =  INF;
                    );
                wmax.FX(           Times  )
                            $(ORD(Times) - 1 = N) = 0;
                qBar.FX(    Queue, Times  )
                             $(ORD(Times) - 1 = N) = 0;
                SOLVE HorizonLP USING LP MINIMIZING Csum;
                SolveTime = HorizonLP.RESUSD;
                Cmax = SUM(TimeInt, dt.L(TimeInt));
                yDot(Machine, TimeInt)
                    = ( dy.L(Machine, TimeInt)
                       /dt.L(         TimeInt))
                       $(dt.L(         TimeInt) > 0);
                uDot(Queue  , TimeInt)
                    = ( du.L(Queue  , TimeInt)
                       /dt.L(         TimeInt))
                       $(dt.L(         TimeInt) > 0);
                LOOP(Times$TimeInt(Times),
                    u(Queue, Times)    = u(   Queue, Times - 1)
                                       + du.L(Queue, Times    );
                    t(       Times)    = t(          Times - 1)
                                       + dt.L(       Times    );
                    );
                v(   Queue  , TimeInt) = uDot(  Queue  , TimeInt)
                                        *p(     Queue)          ;
                w(   Machine, TimePnt) = wmax.L(         TimePnt)
                                       -wDev.L(Machine, TimePnt);
                Ch = SUM(Times$TimeInt(Times),
                    dt.L(Times)*SUM(      Queue,
                                 cBar(    Queue)
```

```
                                    *(  qBar.L(Queue, Times - 1)
                                      + qBar.L(Queue, Times    ))))/2;
                Cw = SUM(Times$TimeInt(Times),
                    dt.L(Times)*(  wmax.L(       Times - 1)
                                      + wmax.L(       Times    )))/2;

$INCLUDE "c:\GAMSCode\FluidOut.gms"

                wmaxBest = SUM(Times$(ORD(Times) - 1 = N - 1),
                                    wmax.L(Times));
                );
            );
        );

    CBest = Csum.L;

    IF ((Method >= 2) AND (Method <= 4),

*       Subdivide each time interval and keep the best.
        CLast = Csum.L;
        cq(Times)$TimePnt(Times)
            = SUM(Queue, cBar(Queue)*qBar.L(Queue, Times));
        BpqBar(Machine, Times)$TimePnt(Times)
            = SUM(Queue, Bp(Machine, Queue)*qBar.L(Queue,Times));
        IF (OutputLevl >= 2,
            DISPLAY cq, BpqBar;
            );

        LOOP (Times$TimeInt(Times),

*           Interpolate time interval for a feasible solution.
            nNow    = ORD(Times) - 1;
            constLP = Csum.L
                    - dt.L(Times)*(  etah*cq(    Times - 1)
                                   + etaw*wmax.L(Times - 1));
            dtCoef  = etah*(  cq(    Times - 1)
                            - cq(    Times    ))
                    + etaw*(  wmax.L(Times - 1)
                            + wmax.L(Times    ));
            qBarCoef(Queue) = etah*cBar(Queue)
                                *dt.L(Times);
            wmaxCoef        = etaw*dt.L(Times);
            dtInterval.UP = dt.L(Times);
            dtInterval.L  = dt.L(Times)/2;
            qBarIntrvl.L(Queue) = (  qBar.L(Queue, Times - 1)
                                   + qBar.L(Queue, Times    ))/2;
            duValue.LO(Queue)   =    qBar.L(Queue, Times    )
                                - alphaBar( Queue)*dt.L(Times);
            duValue.UP(Queue)   =    qBar.L(Queue, Times - 1);
```

```
            duValue.LO(Queue)
*                            $(    duValue.UP(Queue)
*                                 /duValue.LO(Queue)
*                                > OpTolernce       )
                            = MIN(duValue.LO(Queue),
                                  duValue.UP(Queue) );
            duValue.L( Queue)   = -alphaBar(  Queue)*dtInterval.L
                            + qBarIntrvl.L(Queue);
            dyValue.LO(Machine) = BpqBar(Machine, Times - 1);
            dyValue.UP(Machine) = BpqBar(Machine, Times    )
                            + chi(Machine)*dt.L(Times);
            dyValue.UP(Machine)
*                            $(    dyValue.UP(Machine)
*                                 /dyValue.LO(Machine)
*                                > OpTolernce         )
                            = MAX(dyValue.UP(Machine),
                                  dyValue.LO(Machine) );
            dyValue.L( Machine) = chi(Machine)*dtInterval.L
                            + SUM(Queue, Bp(Machine, Queue)
                                       *qBarIntrvl.L(Queue));
            IF (OutputLevl >= 2,
                DISPLAY constLP, dtCoef, qBarCoef, wmaxCoef,
                        dtInterval.L, dtInterval.UP,
                        qBarIntrvl.L,
                        duValue.LO, duValue.L, duValue.UP,
                        dyValue.LO, dyValue.L, dyValue.UP;
                );

*           Optimize this single-time-interval solution with LP.
            SOLVE IntervalLP USING LP MINIMIZING CsumIntrvl;

            IF (OutputLevl >= 2,
                DISPLAY nNow, CsumIntrvl.L,
                        qBarIntrvl.L,
                        wmaxIntrvl.L,
                        dtInterval.L, dtInterval.M,
                        dtInterval.UP,
                        duInterval.L, duInterval.M,
                        dyInterval.L, dyInterval.M,
                         qInterval.L,  qInterval.M,
                        wDevIntrvl.L, wDevIntrvl.M,
                        duValue.LO, duValue.L, duValue.UP,
                        duValue.M,
                        dyValue.LO, dyValue.L, dyValue.UP,
                        dyValue.M;
                );

*           Keep feasible solution if it is best found so far.
            IF (CsumIntrvl.L < CBest,
```

```
                CBest           = CsumIntrvl.L;
                nBest           = ORD(Times) - 1;
                dtBest          = dtInterval.L;
                qBarBest(Queue) = qBarIntrvl.L(Queue);
                wmaxBest        = wmaxIntrvl.L;
                SolveTime       = IntervalLP.RESUSD;
                );

*           Check if LP is good enough.
            IF (N = 1,
                Need4MoreT = dtInterval.M$(  dtInterval.L
                                          = dtInterval.UP)
                           + SUM(Queue, cBar(Queue)
                               *qBarIntrvl.L(Queue)/2
                                     - alphaBar(Queue)
                                      *duValue.M(Queue)
                                 $(  duValue.L(Queue)
                                  = duValue.LO(Queue)))
                           + SUM(Machine, chi(Machine)
                                    *dyValue.M(Machine)
                                $(  dyValue.L(Machine)
                                 = dyValue.UP(Machine)));
                IF (OutputLevl >= 2,
                    DISPLAY CBest, Need4MoreT,
                        dtInterval.M, dtInterval.L,
                        dtInterval.UP,
                        cBar, qBarIntrvl.L, alphaBar, chi,
                        duInterval.L, duInterval.M,
                        duValue.M, duValue.L, duValue.LO,
                        dyInterval.L, dyInterval.M,
                        dyValue.M, dyValue.L, dyValue.UP;
                    );
                qBarCoef(Queue) = etah*cBar(Queue)
                                       *Tstar*1.0001;
                wmaxCoef        = etaw*Tstar*1.0001;
                dtInterval.UP   =      Tstar*1.0001;
                duValue.LO(Queue) = -alphaBar(Queue)
                                     *Tstar*1.0001;
                dyValue.UP(Machine) = chi(Machine)*Tstar*1.0001;
                SOLVE IntervalLP USING LP MINIMIZING CsumIntrvl;
                Need4MoreT = CBest - CsumIntrvl.L;
$ONTEXT
                Need4MoreT = dtInterval.M$(  dtInterval.L
                                          = dtInterval.UP)
                           + SUM(Queue, cBar(Queue)
                               *qBarIntrvl.L(Queue)/2
                                     - alphaBar(Queue)
                                      *duValue.M(Queue)
                                 $(  duValue.L(Queue)
```

```
                                    = duValue.LO(Queue)))
                            + SUM(Machine, chi(Machine)
                                    *dyValue.M(Machine)
                                 $(  dyValue.L(Machine)
                                  = dyValue.UP(Machine)));
$OFFTEXT
                IF (OutputLevl >= 2,
                    DISPLAY CsumIntrvl.L, Need4MoreT,
                        dtInterval.M, dtInterval.L,
                        dtInterval.UP,
                        cBar, qBarIntrvl.L, alphaBar, chi,
                        duInterval.L, duInterval.M,
                        duValue.M, duValue.L, duValue.LO,
                        dyInterval.L, dyInterval.M,
                        dyValue.M, dyValue.L, dyValue.UP;
                    );
                );
            );
        );

    IF (Method = 0,
         CBest = INF;
    ELSEIF (CBest < CLast*OpTolernce) AND (N <= NBound),
        IF (Method <= 4,
            N = N + 1;
        ELSEIF Method = 5,
            N = N*2;
            );
        );
    IF (OutputLevl >= 2,
         DISPLAY Method, CsumCmax, Csum.L, CBest, CLast, N,
                 NBound;
         );
    );
*N = N - 1;

IF (Method = 3,

*   Optimize current solution with full-horizon QP.
    SOLVE HorizonNLP USING NLP MINIMIZING Csum;
    SolveTime = HorizonNLP.RESUSD;
    Cmax = SUM(TimeInt, dt.L(TimeInt));
    yDot(Machine, TimeInt) = ( dy.L(Machine, TimeInt)
                              /dt.L(         TimeInt))
                            $(dt.L(         TimeInt) > 0);
    uDot(Queue  , TimeInt) = ( du.L(Queue  , TimeInt)
                              /dt.L(         TimeInt))
                            $(dt.L(         TimeInt) > 0);
    LOOP(Times$TimeInt(Times),
```

```
        u(Queue, Times)      = u(   Queue, Times - 1)
                             + du.L(Queue, Times    );
        t(       Times)      = t(          Times - 1)
                             + dt.L(       Times    );
        );
    v(   Queue  , TimeInt) = uDot(  Queue  , TimeInt)
                            *p(     Queue)           ;
    w(   Machine, TimePnt) = wmax.L(         TimePnt)
                           - wDev.L(Machine, TimePnt);
    Ch = SUM(Times$TimeInt(Times),
                      dt.L(Times)*SUM(    Queue,
                                  cBar(     Queue)
                                 *(  qBar.L(Queue, Times - 1)
                                   + qBar.L(Queue, Times    ))))/2;
    Cw = SUM(Times$TimeInt(Times),
                      dt.L(Times)*(  wmax.L(     Times - 1)
                                   + wmax.L(     Times    )))/2;
    );

IF (NormMIP >= 0,

*   Optimize start-up solution with MIP.
    yDotHat.UP(Machine) = Idling;
    vHat.UP(Queue)      = q0(Queue);
    vHat.L( Queue)      = v(Queue, 'n1');
    vDev1.L(Queue)      = 0;
    vDevInf.L(Machine)  = 0;
    NormvDev.L          = 0;

    SOLVE StartUpMIP USING MIP MINIMIZING NormvDev;

    IF (OutputLevl >= 2,
        DISPLAY v, vHat.L, vDev1.L, vDevInf.L, NormvDev.L;
        );
    );

$INCLUDE "c:\GAMSCode\FluidOut.gms"

IF (OutputLevl = -1,
*    PUT %gams.seed%
    PUT %gams.user1%
    LOOP(Queues$Queue(Queues),
        PUT c(        Queues);
        );
    LOOP(Queues$Queue(Queues),
        PUT p(        Queues);
        );
    LOOP(Queues$Queue(Queues),
        PUT alpha(    Queues);
```

```
        );
    LOOP(Queues$Queue(Queues),
        PUT q0(       Queues);
        );
    LOOP(Queues$Queue(Queues),
        PUT v(        Queues, 'n1');
        );
    LOOP(Queues$Queue(Queues),
*        PUT vHat.L(   Queues);
        );
    LOOP(Machines$Machine(Machines),
        PUT rho(          Machines);
        );
    LOOP(Machines$Machine(Machines),
        PUT wBar0(        Machines);
        );
    LOOP(Machines$Machine(Machines),
        PUT Tmax(         Machines);
        );
    PUT CsumCmax Csum.L Need4MoreT Tstar Cmax N;
    LOOP(Times$TimeInt(Times),
        PUT t(Times);
        );
    );

PUTCLOSE FluidOut;
```

Appendix B: Simulation Detail

B.1 STATE VARIABLE DEFINITIONS

For this simulation, the following state variables are defined:

`alpha[`*K*`]`	mean external arrival rates for each queue (real)
`alphaBar[`*K*`]`	upstream arrival rates for each queue (real)
`alphaCV[`*K*`]`	arrivals deterministic (0) or exponential (1) (integer)
`Best`	1 if queue is best to draw from (integer)
`beta[`*K*`]`	batch size for each queue (real)
`Busy[`*I*`,`*M*`]`	whether each machine is in service (integer)
`c[`*K*`]`	holding costs for each queue (real)
`Cond`	local generic logical edge condition (integer)
`Csum`	fluid objective value (real)
`Delay`	local time delay (real)
`Done`	local loop-finished logical edge condition (integer)
`dt[`*N*`]`	time period lengths (real)
`ENT[15]`	input and output variable for ranked lists (real)
`eta`	minimize holding cost (0) or maximum workload (1) (real)
`Failures[`*I*`]`	number of failure modes for each machine type (integer)
`i`	index for machine type (integer)
`ICap`	number of machine types (integer)
`idle[`*I*`]`	number of idle machines of each type (integer)
`iota[`*K*`]`	machine type assigned to each queue (integer)
`k`	index for queue (buffer or job stage) (integer)

`kBest`	best queue for a machine to serve next (integer)
`KCap`	number of queues (buffers or job stages) (integer)
`kLast`	last queue that a machine served (integer)
`kNext`	next queue that a machine might serve (integer)
`l`	index for a variety of purposes (integer)
`LastQ[`*I*`,`*M*`]`	last queue that each machine served (integer)
`Lateness`	lag of actual production behind fluid solution (real)
`m`	index for machine number (within a machine type) (integer)
`MCap[`*I*`]`	number of machines of each type (integer)
`MostLate`	actual production most behind fluid solution (real)
`MTTF[`*I*`,5]`	mean time (or runs) to fail in each failure mode (real)
`MTTR[`*I*`,5]`	mean time to repair in each failure mode (real)
`n`	index for current time interval (integer)
`NCap`	number of time intervals in fluid solution (integer)
`NewProb`	reduced random number for probabilistic routing (real)
`nn`	index for time intervals or break-points (integer)
`NumQs[`*I*`]`	number of queues assigned to each machine type (integer)
`p[`*K*`]`	mean machine processing times for each queue (real)
`pCV[`*K*`]`	processing deterministic (0) or exponential (1) (integer)
`Policy`	algorithm for dispatching machines: 0 = Fluid Run-Out, 1 = FIFO, 2 = LIFO, 3 = FBFS, 4 = LBFS, 5 = Priority Given, 6 = Planar (`ICap` = 2 & `KCap` = 3 only) (integer)
`Pr[`*K*`,`*K*`]`	probabilistic (from-to) routing matrix (real)

`Prob`	random number for probabilistic routing (real)
`ProbProc`	processing times deterministic (0) or random (1) (real)
`ProbRout`	routing deterministic (0) or random (1) (integer)
`Priority[`I`,`K`]`	order for queues assigned to each machine type (integer)
`pTran[`K`]`	transit times between machines for each queue (real)
`q[`K`]`	number of jobs in queue, process, or transit (integer)
`q0[`K`]`	initial number of jobs in queue (integer)
`q1Coeff`	coefficient of q_1 for boundary in planar method (real)
`q2Interc`	q_2-axis intercept for boundary in planar method (real)
`q3Coeff`	coefficient of q_3 for boundary in planar method (real)
`qBound`	upper limit on `qTotl` (integer)
`qProc[`K`]`	number of jobs in process on a machine (integer)
`qTotl`	total number of jobs in system (integer)
`qTran[`K`]`	number of jobs in transit between machines (integer)
`qWait[`K`]`	number of jobs waiting in queue for a machine (integer)
`RNK[198]`	rank list on this element of `ENT` (integer)
`RunFails[`I`]`	number of cycle-based machine failure modes (integer)
`Runs[`I`,`M`]`	number of times in row machine served same buffer (integer)
`S[`K`,`K`]`	matrix of sequence-dependent set-up times (real)
`SetUp[`I`]`	if machine type has sequence-dependent set-ups (integer)
`t[`N`]`	time break-points (real)
`t0`	initial time (begins first time interval) (real)
`TCap`	time horizon beyond which arrivals cease (real)

u[*K*,*N*]	fluid number of units produced by break-points (real)
u0[*K*]	initial number of units produced (real)
uDiff[*K*]	difference in number of units produced (real)
uDot[*K*,*N*]	processing rates during each time interval (real)
uHat[*K*]	actual number of units produced by current time (integer)
upsilon	set-up time multiplier for dispatching (real)
uTilde[*K*]	fluid number of units produced by current time (real)
Variate	random-number antithetic variates to use (0 or 1) (integer)
vHat[*K*]	initial number of machines working each queue (integer)

B.2 EVENT DEFINITIONS

1. The Size(Variate) event occurs when the problem size and general variables are read. Initial values for Variate are needed for each run. This event causes the following state changes:

```
Policy   = DISK{SimIn.XLX; 0},
qBound   = DISK{SimIn.XLX; 0},
q2Interc = DISK{SimIn.XLX; 0},
q1Coeff  = DISK{SimIn.XLX; 0},
q3Coeff  = DISK{SimIn.XLX; 0},
upsilon  = DISK{SimIn.XLX; 0},
Csum     = DISK{SimIn.XLX; 0},
eta      = DISK{SimIn.XLX; 0},
ICap     = DISK{SimIn.XLX; 0},
KCap     = DISK{SimIn.XLX; 0},
```

```
ProbProc = DISK{SimIn.XLX; 0},
ProbRout = DISK{SimIn.XLX; 0},
NCap     = DISK{SimIn.XLX; 0},
TCap     = DISK{SimIn.XLX; 0},
t0       = DISK{SimIn.XLX; 0},
t[0]     = t0,
qTotl    = 0,
n        = 1,
ENT[0]   = 1
```

After every occurrence of the `Size` event:

If `NCap` > 0, then read time break points from a fluid solution; that is, schedule the `tOpt(nn)` event to occur without delay using the parameter value of 1. (Time ties are broken by an execution priority of 1.)

If `NCap` > 0, then read time interval lengths from a fluid solution; that is, schedule the `dtOpt(nn)` event to occur without delay using the parameter value of 1. (Time ties are broken by an execution priority of 3.)

Always read machine information; that is, schedule the `Tools(i)` event to occur without delay using the parameter value of 1.

Always read buffer information; that is, schedule the `Queues(k)` event to occur in `t0` time units using the parameter value of 1. (Time ties are broken by an execution priority of 9.)

2. The `tOpt(nn)` event occurs when the time break points from the fluid model are read. This event causes the following state change:

`t[nn] = DISK{SimIn.XLX; 0}`

After every occurrence of the `tOpt` event:

If `nn < NCap`, then read the next time break point from a fluid solution; that is, schedule the `tOpt(nn)` event to occur without delay using the parameter value of `nn + 1`. (Time ties are broken by an execution priority of 2.)

3. The `dtOpt(nn)` event occurs when the interval lengths from the fluid model are read. This event causes the following state change:

`dt[nn] = DISK{SimIn.XLX; 0}`

After every occurrence of the `dtOpt` event:

If `nn < NCap`, then read the next time interval length from a fluid solution; that is, schedule the `dtOpt(nn)` event to occur without delay using the parameter value of `nn + 1`. (Time ties are broken by an execution priority of 4.)

4. The `Tools(i)` event occurs when the data for each machine type are read. This event causes the following state changes:

```
MCap[i]       = DISK{SimIn.XLX; 0},
idle[i]       = MCap[i],
SetUp[i]      = DISK{SimIn.XLX; 0},
NumQs[i]      = DISK{SimIn.XLX; 0},
Failures[i]   = DISK{SimIn.XLX; 0},
```

```
RunFails[i]   = DISK{SimIn.XLX; 0},
RNK[KCap + i] = 0
```

After every occurrence of the `Tools` event:

If `Failures[i] > 0`, then read machine failure information; that is, schedule the `Failure(i, l)` event to occur without delay using the parameter values of `i`, 1. (Time ties are broken by an execution priority of 6.)

If `i < ICap`, then read machine information for the next type of machine; that is, schedule the `Tools(i)` event to occur without delay using the parameter value of `i + 1`. (Time ties are broken by an execution priority of 8.)

5. The `Failure(i, l)` event occurs when the failure and repair data for each machine type are read. This event causes the following state changes:

```
MTTF[i; l] = DISK{SimIn.XLX; 0},
MTTR[i; l] = DISK{SimIn.XLX; 0}
```

After every occurrence of the `Failure` event:

If `l < Failures[i]`, then read machine failure information for the next type of failure; that is, schedule the `Failure(i, l)` event to occur without delay using the parameter values of `i, l + 1`. (Time ties are broken by an execution priority of 7.)

6. The `Queues(k)` event occurs when the data for each buffer are read. This event causes the following state changes:

```
alpha[k]          = DISK{SimIn.XLX; 0},
```

```
alphaBar[k]       = DISK{SimIn.XLX; 0},
alphaCV[k]        = DISK{SimIn.XLX; 0},
beta[k]           = DISK{SimIn.XLX; 0},
c[k]              = DISK{SimIn.XLX; 0},
iota[k]           = DISK{SimIn.XLX; 0},
kNext             = DISK{SimIn.XLX; 0},
Priority[iota[k];
         kNext  ] = k,
p[k]              = DISK{SimIn.XLX; 0},
pCV[k]            = DISK{SimIn.XLX; 0},
pTran[k]          = DISK{SimIn.XLX; 0},
q0[k]             = DISK{SimIn.XLX; 0},
vHat[k]           = DISK{SimIn.XLX; 0},
u0[k]             = DISK{SimIn.XLX; 0},
RNK[k]            = 0,
qTotl             = qTotl + q0[k],
u[k; 0]           = u0[k],
uTilde[k]         = u0[k],
uHat[k]           = u0[k],
uDiff[k]          = 0,
uDot[k; NCap + 1] = alphaBar[k]
```

After every occurrence of the `Queues` event:

If `NCap` > 0, then read the number of units produced by time break-points from a fluid solution; that is, schedule the `uOpt(k, nn)` event to occur without delay using the parameter values of `k, 1`. (Time ties are broken by an execution priority of 10.)

If `NCap` > 0, then read the processing rates during each time interval from a fluid solution; that is, schedule the `uDotOpt(k, nn)` event to occur without delay using the parameter values of `k, 1`. (Time ties are broken by an execution priority of 12.)

Always read the routing information; that is, schedule the `Routes(k, kNext)` event to occur without delay using the parameter values of `k, 1`. (Time ties are broken by an execution priority of 14.)

Always read the information on sequence-dependent setups; that is, schedule the `SetUps(k, kNext)` event to occur without delay using the parameter values of `k, 1`. (Time ties are broken by an execution priority of 16.)

If `q0[k]` > 0, then allocate the initial WIP; that is, schedule the `Initial(k, 1)` event to occur without delay using the parameter values of `k, 1`. (Time ties are broken by an execution priority of 18.)

If `alpha[k] > 0`, then order an arrival stream; that is, schedule the `Arrive(k)` event to occur without delay using the parameter value of `k`. (Time ties are broken by an execution priority of 24.)

If `k < KCap`, then read information for the next buffer type; that is, schedule the `Queues(k)` event to occur without delay using the parameter value of `k + 1`. (Time ties are broken by an execution priority of 25.)

7. The `uOpt(k, nn)` event occurs when the unit output data for each buffer from the fluid model are read. This event causes the following state change:

`u[k; nn] = DISK{SimIn.XLX; 0}`

After every occurrence of the `uOpt` event:

If `nn < NCap`, then read the number of units produced by the next time break-point from a fluid solution; that is, schedule the `uOpt(k, nn)` event to occur without delay using the parameter values of `k, nn + 1`. (Time ties are broken by an execution priority of 11.)

8. The `uDotOpt(k, nn)` event occurs when the unit output rate data for each buffer from the fluid model are read. This event causes the following state change:

`uDot[k; nn] = DISK{SimIn.XLX; 0}`

After every occurrence of the `uDotOpt` event:

If `nn < NCap`, then read the processing rates during the next time interval from a fluid solution; that is, schedule the `uDotOpt(k, nn)` event to occur without delay using the parameter values of `k, nn + 1`. (Time ties are broken by an execution priority of 13.)

9. The `Routes(k, kNext)` event occurs when the routing probabilities between each pair of buffers are read. This event causes the following state change:

`Pr[k; kNext] = DISK{SimIn.XLX; 0}`

After every occurrence of the Routes event:

If `kNext < KCap`, then read the probability of moving to the next buffer; that is, schedule the `Routes(k, kNext)` event to occur without delay using the parameter values of `k, kNext + 1`. (Time ties are broken by an execution priority of 15.)

10. The `SetUps(k, kNext)` event occurs when the setup times between each pair of buffers are read. This event causes the following state change:

`S[k; kNext] = DISK{SimIn.XLX; 0}`

After every occurrence of the `SetUps` event:

If `kNext < KCap`, then read the sequence-dependent setup required if changing to the next buffer; that is, schedule the `SetUps(k, kNext)` event to occur without delay using the parameter values of `k, kNext + 1`. (Time ties are broken by an execution priority of 17.)

11. The `Initial(k, l)` event occurs when the beginning queue contents are filled. This event causes the following state change:

`Cond = (l <= vHat[k]*beta[k])`

After every occurrence of the `Initial` event:

If `l < q0[k]`, then allocate the next unit of initial WIP; that is, immediately execute the `Initial(k, l)` event using the parameter values of `k, l + 1`. (Time ties are broken by an execution priority of 19.)

If `ProbProc`, then read in the random numbers for processing times; that is, immediately execute the `PutRand(kNext)` event using the parameter value of 1. (Time ties are broken by an execution priority of 20.)

If `Cond`, then allocate an initial WIP unit that starts processing immedately; that is, schedule the `NexType(k)` event to occur without delay using the parameter value of `k`. (Time ties are broken by an execution priority of 22.)

If `Cond == 0`, then allocate an initial WIP unit that waits in a buffer; that is, schedule the `NexType(k)` event to occur without delay using the parameter value of `k`. (Time ties are broken by an execution priority of 23.)

12. The `Arrive(k)` event occurs when arrivals are scheduled. This event causes the following state changes:

```
Cond   = (CLK > t0)
       & (qTotl < qBound),
qTotl  = qTotl + Cond,
ENT[0] =      1 -   Variate
       - RND*(1 - 2*Variate),
Delay  = (1 - alphaCV[k]
             *(1 + LN{ENT[0]}))
        /alpha[k]
```

After every occurrence of the `Arrive` event:

If Cond and ProbProc, then read in random numbers for processing times; that is, immediately execute the PutRand(kNext) event using the parameter value of 1. (Time ties are broken by an execution priority of 26.)

If CLK + Delay < t0 + TCap, then schedule the next arrival; that is, schedule the Arrive(k) event to occur in Delay time units using the parameter value of k. (Time ties are broken by an execution priority of 27.)

If Cond, then allocate the arrival to a buffer; that is, schedule the NexType(k) event to occur without delay using the parameter value of k. (Time ties are broken by an execution priority of 28.)

13. The PutRand(kNext) event occurs when random numbers associated with this arrival's processing times are stored. This event causes the following state changes:

```
ENT[0] =      1 -   Variate
       - RND*(1 - 2*Variate),
ENT[0] = PUT{FIF; kNext}
```

After every occurrence of the PutRand event:

If kNext < KCap, then read in next random number for processing times; that is, immediately execute the PutRand(kNext) event using the parameter value of kNext + 1. (Time ties are broken by an execution priority of 21.)

14. The NexType(k) event occurs when the next machine type is looked up and the queues are incremented. This event causes the following state changes:

```
i         = iota[k],
q[k]      = q[k]     + 1,
qWait[k]  = qWait[k] + 1,
Cond      = (idle[i] > 0)
          & (qWait[k] >= beta[k])
```

After every occurrence of the `NexType` event:

If (0 < `Policy`) and (`Policy` < 5), then insert the unit in a ranked list; that is, immediately execute the `PutQ(i, k)` event using the parameter values of `i, k`. (Time ties are broken by an execution priority of 29.)

If `Cond`, then allocate the unit to a specific machine; that is, immediately execute the `NexTool(i, k, m)` event using the parameter values of `i, k, 1`. (Time ties are broken by an execution priority of 30.)

15. The `PutQ(i, k)` event occurs when the unit's buffer number is stored in a ranked list. This event causes the following state changes:

```
ENT[0] = k,
ENT[0] = PUT{Policy; KCap + i}
```

No additional events are scheduled here.

16. The `NexTool(i, k, m)` event occurs when the next machine is seized. This event causes the following state change:

```
Cond = Busy[i; m] & (m < MCap[i])
```

After every occurrence of the `NexTool` event:

If `Cond`, then check the availability of the next specific machine; that is, immediately execute the `NexTool(i, k, m)` event using the parameter values of `i, k, m + 1`. (Time ties are broken by an execution priority of 31.)

If `Busy[i; m] == 0`, then have the machine begin setup (if any) and processing; that is, immediately execute the `Start(i, kLast, k, m)` event using the parameter values of `i, LastQ[i; m], k, m`. (Time ties are broken by an execution priority of 32.)

17. The `Start(i, kLast, k, m)` event occurs when processing begins. This event causes the following state changes:

```
Busy[i; m] = 1 - ProbProc + GET{FST; k},
idle[i]    = idle[i] - Busy[i; m],
Runs[i; m] = Runs[i; m]*(k == kLast)
           + beta[k]*Busy[i; m],
qWait[k]   = qWait[k]
           - beta[k]*Busy[i; m],
qProc[k]   = qProc[k]
           + beta[k]*Busy[i; m],
Delay      = S[kLast; k]
           + (1 - pCV[k]*(1 + LN{ENT[0]}))
            *p[k]
```

After every occurrence of the `Start` event:

If Busy[i; m], then schedule the completion of processing; that is, schedule the Repair(i, k, m, l, Delay) event to occur in Delay time units using the parameter values of i, k, m, 1, Delay. (Time ties are broken by an execution priority of 33.)

18. The Repair(i, k, m, l, Delay) event occurs when failures (if any) are accounted for.

After every occurrence of the Repair event:

If l < RunFails[i], then schedule the completion of this run-dependent repair (if any); that is, schedule the Repair(i, k, m, l, Delay) event to occur in -LN{RND}*MTTR[i; l]*(RND*MTTF[i; l] < 1) time units using the parameter values of i, k, m, l + 1, Delay. (Time ties are broken by an execution priority of 34.)

If (RunFails[i] <= l) and (l < Failures[i]), then schedule the completion of this time-dependent repair (if any); that is, schedule the Repair(i, k, m, l, Delay) event to occur in -LN{RND}*MTTR[i; l]*(-LN{RND}*MTTF[i; l] < Delay) time units using the parameter values of i, k, m, l + 1, Delay. (Time ties are broken by an execution priority of 35.)

If l >= Failures[i], then finish repairs; that is, schedule the Finish(i, k, m) event to occur without delay using the parameter values of i, k, m. (Time ties are broken by an execution priority of 36.)

19. The `Finish(i, k, m)` event occurs when processing ends. This event causes the following state changes:

```
idle[i]     = idle[i] + 1,
Busy[i; m]  = 0,
LastQ[i; m] = k,
qProc[k]    = qProc[k] - beta[k],
qTran[k]    = qTran[k] + beta[k]
```

After every occurrence of the `Finish` event:

Always disaggregate batches (if any); that is, schedule the `UnBatch(k, l)` event to occur without delay using the parameter values of `k, 1`. (Time ties are broken by an execution priority of 37.)

If `Policy` == 0, then determine the current time interval from the fluid solution; that is, immediately execute the `Nextn(i, k, m, nn)` event using the parameter values of `i, k, m, n - 1`. (Time ties are broken by an execution priority of 45.)

If (0 < `Policy`) and (`Policy` < 5), then get the next unit from the ranked list; that is, immediately execute the `NexQRnk(i, k, m)` event using the parameter values of `i, k, m`. (Time ties are broken by an execution priority of 53.)

If `Policy` == 5, then determine if enough units have been processed from this buffer; that is, immediately execute the `ChangeQ(i, k, m)` event using the parameter values of `i, k, m`. (Time ties are broken by an execution priority of 55.)

If `Policy` == 6, then determine the current side of decision boundary (in planar method); that is, immediately execute the `NexQPln(i, k, m)` event using the parameter values of `i, k, m`. (Time ties are broken by an execution priority of 60.)

20. The `UnBatch(k, l)` event occurs when process batches are disassociated.
After every occurrence of the `UnBatch` event:
If `l < beta[k]`, then unpack the next unit from the batch; that is, schedule the `UnBatch(k, l)` event to occur without delay using the parameter values of `k, l + 1`. (Time ties are broken by an execution priority of 38.)
If `ProbRout`, then probabilistically determine the next buffer to which the unit will be sent; that is, schedule the `NexStep(k, kNext, Prob)` event to occur without delay using the parameter values of `k, 1, RND`. (Time ties are broken by an execution priority of 39.)
If `ProbRout` == 0, then deterministically determine the next buffer to which the unit will be sent; that is, schedule the `NexStep(k, kNext, Prob)` event to occur without delay using the parameter values of `k, 1, 0.5`. (Time ties are broken by an execution priority of 40.)

21. The `NexStep(k, kNext, Prob)` event occurs when "dice" are rolled to find which queue (if any) to go to next. This event causes the following state changes:

```
NewProb = Prob - Pr[k; kNext],
```

```
Done    = (kNext == KCap),
Cond    = (NewProb > 0)
```

After every occurrence of the `NexStep` event:

If (`Done` == 0) and `Cond`, then see if the next buffer is the right one to which the unit should be sent; that is, schedule the `NexStep(k, kNext, Prob)` event to occur without delay using the parameter values of `k, kNext + 1, NewProb`. (Time ties are broken by an execution priority of 41.)

If `Cond` == 0, then transport the unit to the next buffer; that is, schedule the `Transit(k, kNext)` event to occur in `pTran[k]` time units using the parameter values of `k, kNext`. (Time ties are broken by an execution priority of 42.)

If `Done` and `Cond`, then transport the unit out of the factory; that is, schedule the `Transit(k, kNext)` event to occur in `pTran[k]` time units using the parameter values of `k, 0`. (Time ties are broken by an execution priority of 43.)

22. The `Transit(k, kNext)` event occurs when units arrive at the next queue. This event causes the following state changes:

```
qTran[k] = qTran[k] - 1,
q[k]     = q[k]     - 1,
qTotl    = qTotl    - (kNext == 0)
```

After every occurrence of the `Transit` event:

If `kNext` > 0, then put the unit in its next buffer; that is, schedule the `NexType(k)` event to occur without delay using the parameter value of `kNext`. (Time ties are broken by an execution priority of 44.)

23. The `Nextn(i, k, m, nn)` event occurs when the current time interval is determined. This event causes the following state changes:

```
nn   = nn + 1,
Done = (nn == NCap + 1) | (CLK <= t[nn])
```

After every occurrence of the `Nextn` event:

If `Done` == 0, then try the next time interval from the fluid solution; that is, immediately execute the `Nextn(i, k, m, nn)` event using the parameter values of `i, k, m, nn`. (Time ties are broken by an execution priority of 46.)

If `Done`, then measure the lag in this buffer's current unit output versus the fluid model; that is, immediately execute the `Compare(i, k, m, n)` event using the parameter values of `i, k, m, nn`. (Time ties are broken by an execution priority of 47.)

24. The `Compare(i, k, m, n)` event occurs when the lag in this buffer's current unit output versus the fluid model is measured. This event causes the following state changes:

```
uTilde[k] = u[k; n - 1]
          + uDot[k; n]*(CLK - t[n - 1]),
uDiff[k]  = uTilde[k] - uHat[k],
```

```
uHat[k]    = uHat[k] + beta[k]
```

After every occurrence of the `Compare` event:

Always measure the lag in other buffers' current unit output versus the fluid model; that is, immediately execute the `NexQOpt(i, kLast, m, l, kBest, MostLate)` event using the parameter values of `i, k, m, 1, 0, -99999`. (Time ties are broken by an execution priority of 48.)

25. The `NexQOpt(i, kLast, m, l, kBest, MostLate)` event occurs when the lag in other buffers' current unit outputs versus the fluid model are measured. This event causes the following state changes:

```
k          = Priority[i; l],
uTilde[k] = u[k; n - 1]
           + uDot[k; n]*(CLK - t[n - 1]),
uDiff[k]   = uTilde[k] - uHat[k],
Lateness   = uDiff[k]
           - uDot[k; n]*upsilon*S[kLast; k],
Best       = (Lateness > MostLate)
           & (qWait[k] >= beta[k]),
Done       = (l == NumQs[i])
```

After every occurrence of the `NexQOpt` event:

If (`Done` == 0) and (`Best` == 0), then look at the next buffer on the priority list; that is, immediately execute the `NexQOpt(i, kLast, m, l, kBest,`

MostLate) event using the parameter values of i, kLast, m, l + 1, kBest, MostLate. (Time ties are broken by an execution priority of 49.)

If (Done == 0) and Best, then look at the next buffer on the priority list (keeping the this one as the best so far); that is, immediately execute the NexQOpt(i, kLast, m, l, kBest, MostLate) event using the parameter values of i, kLast, m, l + 1, k, Lateness. (Time ties are broken by an execution priority of 50.)

If Done and Best, then begin processing from this buffer; that is, immediately execute the Start(i, kLast, k, m) event using the parameter values of i, kLast, k, m. (Time ties are broken by an execution priority of 51.)

If Done and (Best == 0) and kBest, then begin processing from the best buffer so far; that is, immediately execute the Start(i, kLast, k, m) event using the parameter values of i, kLast, kBest, m. (Time ties are broken by an execution priority of 52.)

26. The NexQRnk(i, k, m) event occurs when the next buffer (for the machine to serve) is pulled from the ranked list. This event causes the following state changes:

```
Cond  = GET{FST; KCap + i},
kNext = ENT[0],
Cond  = (iota[kNext] == i)
      & (qWait[kNext] > 0)
```

After every occurrence of the NexQRnk event:

If Cond, then begin processing this unit; that is, immediately execute the Start(i, kLast, k, m) event using the parameter values of i, k, kNext, m. (Time ties are broken by an execution priority of 54.)

27. The ChangeQ(i, k, m) event occurs when a change of buffers is considered. This event causes the following state change:

```
Cond = SetUp[i]
     & (Runs[i; m] < upsilon)
     & (qWait[k] >= beta[k])
```

After every occurrence of the ChangeQ event:

If Cond, then begin processing from this buffer; that is, immediately execute the Start(i, kLast, k, m) event using the parameter values of i, k, k, m. (Time ties are broken by an execution priority of 56.)

If Cond == 0, then determine the next buffer on the priority list from which to process; that is, immediately execute the NexQPri(i, kLast, m, l) event using the parameter values of i, k, m, 1. (Time ties are broken by an execution priority of 57.)

28. The NexQPri(i, kLast, m, l) event occurs when the highest priority non-empty buffer is chosen. This event causes the following state changes:

```
k    = Priority[i; l],
Best = (qWait[k] >= beta[k])
```

After every occurrence of the NexQPri event:

If (`l < NumQs[i]`) and (`Best == 0`), then check the next buffer; that is, immediately execute the `NexQPri(i, kLast, m, l)` event using the parameter values of `i, kLast, m, l + 1`. (Time ties are broken by an execution priority of 58.)

If `Best`, then begin processing from this buffer; that is, immediately execute the `Start(i, kLast, k, m)` event using the parameter values of `i, kLast, k, m`. (Time ties are broken by an execution priority of 59.)

29. The `NexQPln(i, k, m)` event occurs when the current side of decision boundary (in the planar method) is determined. This event causes the following state changes:

```
Cond  = (q[2] <  q2Interc + q1Coeff*q[1]
                          + q3Coeff*q[3]),
kNext = 2*((i == 2) & (   qWait[2] >  0))
      + 1*((i == 1) & (   qWait[1] >  0)
                    & (  (qWait[3] == 0)
                       |  Cond          ))
      + 3*((i == 1) & (   qWait[3] >  0)
                    & (  (qWait[1] == 0)
                       | (Cond     == 0)))
```

After every occurrence of the `NexQPln` event:

If `kNext` > 0, then begin processing from the chosen buffer; that is, immediately execute the `Start(i, kLast, k, m)` event using the parameter values of `i, k, kNext, m`. (Time ties are broken by an execution priority of 61.)

Appendix C: Notation

C.1 Typeface

Italic ≡ scalar variable

Bold ≡ column vector (if lower case) or matrix (if upper case) variable

C.2 Sets

$\mathbb{N}$ ≡ positive integers
= $\{1,2,3,\ldots\}$

$\mathbb{R}$ ≡ real numbers

$\mathbb{S}$ ≡ arbitrary set

$\mathbb{Z}$ ≡ integers
= $\{\ldots,-2,-1,0,1,2,\ldots\}$

$\mathbb{Z}_+$ ≡ non-negative integers
= $\{0,1,2,\ldots\}$
= $\mathbb{N} \cup \{0\}$

C.3 Overscripts

$\geq$ ≡ vector of nonnegative decision variables

$\leq$ ≡ inequality constraint matrix or right-hand-side vector

$-$ ≡ infinitely repeated decimal digit or active inequality constraint matrix or right-hand-side vector

$<$ ≡ inactive inequality constraint matrix or right-hand-side vector

$=$ ≡ equality constraint matrix or right-hand-side vector

$\sim$ ≡ fluid solution

$\wedge$ ≡ feasible discrete solution adapted from fluid solution

$\cup$ ≡ deviation

$\cdot$ ≡ derivative with respect to time

C.4 SUPERSCRIPTS

number ≡ power exponent

$'$ ≡ alternate variable of same type (not derivative or matrix transpose)

T ≡ matrix transpose

-1 ≡ inverse of function or matrix

$+$ ≡ add an infinitesimal amount (on a scalar variable) or pre-multiply by $\left(\mathbf{I}_K - \mathbf{P}\right)$ or $\left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1}$ (on a vector variable)

$*$ ≡ optimal solution

C.5 SUBSCRIPTS (MOSTLY DISCRETE INDICES)

i ≡ machine (or tool) type

j ≡ job (or product) type

k ≡ stage (or task or step) in which a job is processing or is waiting in queue

l ≡ order of job in queue

m ≡ machine number

max ≡ maximum over index set

n ≡ time step or interval number

$\mathbf{Q}$ ≡ submatrix corresponding to $q_{j,k}(t_n)$ variables in QP and LP formulations

$\mathbf{Q}^+$ ≡ submatrix corresponding to $q^+_{j,k}(t_n)$ variables in QP and LP formulations

$\Delta\mathbf{Q}^+$ ≡ submatrix corresponding to $\Delta q^+_{j,k}(t_n)$ variables in QP and LP formulations

$\Delta\mathbf{t}$ ≡ submatrix corresponding to Δt_n variables in QP and LP formulations

$\Delta\mathbf{U}$ ≡ submatrix corresponding to $\Delta u_{j,k}(t_n)$ variables in QP and LP formulations

$\Delta\mathbf{Y}$ ≡ submatrix corresponding to $\Delta y_i(t_n)$ variables in QP and LP formulations

C.6 FUNCTION ARGUMENTS (CONTINUOUS INDICES)

t ≡ time

C.7 FUNCTIONS

$\|\mathbf{x}\|$ ≡ norm of vector variable **x**

$|||\mathbf{X}|||$ ≡ norm of matrix variable **X**

$\lfloor\cdot\rfloor$ ≡ floor function which has a value of the greatest integer less than or equal to its real-valued argument

$\langle\cdot\rangle$ ≡ indicator function which has a value of one if its logical argument is true and zero otherwise

C.8 LOGICAL OPERATORS

$\wedge$ ≡ binary And

$\vee$ ≡ binary Or

C.9 HINDU-ARABIC SYMBOLS

C.9.1 Data Variables

$\mathbf{0}$ ≡ vector or matrix of all zeros with dimensions given by context (such as $\mathbf{x} \geq \mathbf{0}$) or by subscript (such as $\mathbf{0}_{1,K}$)

$\mathbf{1}$ ≡ vector of all ones with dimension given by context (such as $\mathbf{1}^{\mathrm{T}}\mathbf{x}$) or by subscript (such as $\mathbf{1}_K$)

C.10 GREEK SYMBOLS

For the convenience of the reader, the Greek alphabet is given below:

$\alpha\beta\gamma\delta\varepsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi o\pi\rho\sigma\tau\upsilon\phi\chi\psi\omega$
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ

C.10.1 Data Variables

$\alpha_{j,k}$ ≡ (alpha) mean external arrival rate of jobs of type *j* to stage *k*

$\alpha^{+}_{j,k}$ ≡ upstream arrival rate (sum of mean external arrival rate of jobs of type *j* to stage *k* or earlier)

$$= \sum_{k'=1}^{k} \alpha_{j,k'}$$

$\boldsymbol{\alpha}$ ≡ vector of external arrival rates where element number $\kappa(j,k)$ has the value $\alpha_{j,k}$

$\boldsymbol{\alpha}^{+}$ ≡ vector of upstream arrival rates where element number $\kappa(j,k)$ has the value $\alpha^{+}_{j,k}$

$$= \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \boldsymbol{\alpha}$$

$\beta_{j,k}$ ≡ (beta) number of jobs of type *j* in stage *k* in process batch size

Δ ≡ (delta) difference operator with respect to adjacent time intervals

η ≡ (eta) starvation avoidance parameter (varies relative weight of C_{h} and C_{w} in objective function)

$\theta_i(t)$ ≡ (theta) machine congestion (mean fraction of time required for machine i to process all of its tasks that are either present at time t_0 or that arrive by time t)

$$= \frac{\sum_{(j,k)\in\sigma_i} p_{j,k}\left[\underline{q}_{j,k}(t_0) + \alpha^{+}_{j,k}\left(t - t_0\right)\right]}{a_i M_i\left(t - t_0\right)}$$

$\theta_{\max}(t)$ ≡ bottleneck congestion

$$= \max\left\{\theta_i(t) : i = 1, 2, \ldots, I\right\}$$

$\iota(j,k)$ ≡ (iota) machine type that processes jobs of type j in stage k

$\kappa(j,k)$ ≡ (kappa) cumulative stage index for jobs of type j in stage k

$$= k + \sum_{j'=1}^{j-1} K_{j'}$$

$\boldsymbol{\lambda}$ ≡ (lambda) vector of Lagrange multipliers in QP and LP formulations (not upstream arrival rate)

$\mu_{j,k}$ ≡ (mu) processing rate for jobs of type j in stage k

$$= \frac{1}{p_{j,k}}$$

ν ≡ (nu) scale factor

ρ_i ≡ (rho) traffic intensity at machine i

$$= \frac{\sum_{(j,k)\in\sigma_i} p_{j,k}\alpha^{+}_{j,k}}{a_i M_i}$$

$\boldsymbol{\rho}$ ≡ vector of traffic intensities

$$= \left[\mathbf{D(a)D(m)}\right]^{-1}\mathbf{BD(p)}\underline{\boldsymbol{\alpha}}$$

σ_i ≡ (sigma) set of job types and stages that are processed on machines of type i

$$= \left\{(j,k) : \iota(j,k) = i\right\}$$

$\left|\sigma_i\right|$ ≡ number of stages that are processed on machines of type i

$\sigma_{\max}$ ≡ largest number of stages that are processed on any machine type

$$= \max\left\{\left|\sigma_i\right| : i = 1,2,\ldots,I\right\}$$

$\tau_{j,k,m}(t_0)$ ≡ (tau) initial amount of service time that machine number m of type $\iota(j,k)$ has spent processing jobs of type j at stage k

$\tau_{j,k}(t_0)$ ≡ initial amount of service time that machines of type $\iota(j,k)$ spent processing jobs of type j at stage k

$$= \sum_{m=1}^{M_{\iota(j,k)}} \tau_{j,k,m}(t_0)$$

υ ≡ (upsilon) algorithm parameter for avoiding sequence-dependent setup times

ϕ_i ≡ (phi) work load of machine type i for one job of each job type

$$= \sum_{(j,k)\in\sigma_i} p_{j,k}$$

$\phi_{\max}$ ≡ largest single-job work load over all machines

= $\max\{\phi_i : i = 1, 2, \ldots, I\}$

χ_i ≡ (chi) discretionary availability of machine of type *i* (average number of machines available but not needed to process ongoing upstream arrivals)

= $a_i M_i - \sum_{(j,k)\in\sigma_i} p_{j,k}\alpha^+_{j,k}$

$\boldsymbol{\chi}$ ≡ vector of discretionary machine availabilities (where element number *i* has the value χ_i)

= $\mathbf{D(a)m - BD(p)}\underline{\boldsymbol{\alpha}}$

ψ_i ≡ (psi) longest individual mean processing time required for machines of type *i*

= $\max\{p_{j,k} : (j,k) \in \sigma_i\}$

$\psi_{\max}$ ≡ longest of all individual mean processing times

= $\max\{\psi_i : i = 1, 2, \ldots, I\}$

C.10.2 Decision Variables

$\gamma_{j,k}(t)$ ≡ (gamma) dual variable that is complementarily slack with $q_{j,k}(t)$

$\boldsymbol{\gamma}(t)$ ≡ vector of dual variables that are complementarily slack with $\mathbf{q}(t)$ where element number $\kappa(j,k)$ has the value $\gamma_{j,k}(t)$

$\boldsymbol{\Gamma}$ ≡ (gamma) matrix of dual variables that are complementarily slack with $\mathbf{Q}$

= $[\boldsymbol{\gamma}(t_1) \quad \boldsymbol{\gamma}(t_2) \quad \cdots \quad \boldsymbol{\gamma}(t_{N-1})]$ (8.16)

δ_n ≡ (delta) dual variable that is complementarily slack with Δt_n

$\boldsymbol{\delta}$ ≡ vector of dual variables that are complementarily slack with $\Delta\mathbf{t}$ where element number n has the value δ_n

$\xi_i(t)$ ≡ (xi) dual variable that is complementarily slack with $\dot{y}_i(t)$ and $\Delta y_i(t)$ (marginal price for average number of available machines of type i at time t)

$\boldsymbol{\xi}(t)$ ≡ vector of dual variables that are complementarily slack with $\dot{\mathbf{y}}(t)$ and $\Delta\mathbf{y}(t)$ where element number i has the value $\xi_i(t)$

$\boldsymbol{\Xi}$ ≡ (xi) matrix of dual variables that are complementarily slack with $\dot{\mathbf{Y}}$ and $\Delta\mathbf{Y}$

$= \left[\boldsymbol{\xi}(t_1) \quad \boldsymbol{\xi}(t_2) \quad \cdots \quad \boldsymbol{\xi}(t_N)\right]$

$\pi_{j,k}(t)$ ≡ (pi) unrestricted dual variable

$\boldsymbol{\pi}(t)$ ≡ vector of unrestricted dual variables where element number $\kappa(j,k)$ has the value $\pi_{j,k}(t)$

$\boldsymbol{\Pi}$ ≡ (pi) matrix of unrestricted dual variables

$= \left[\boldsymbol{\pi}(t_1) \quad \boldsymbol{\pi}(t_2) \quad \cdots \quad \boldsymbol{\pi}(t_N)\right]$

$\tau_{j,k,m}(t)$ ≡ (tau) cumulative equivalent amount of service time that machine number m of type $\iota(j,k)$ has spent processing jobs of type j at stage k by time t

$\tau_{j,k}(t)$ ≡ cumulative equivalent amount of service time that machines of type $\iota(j,k)$ spent processing jobs of type j at stage k by time t

$$= \sum_{m=1}^{M_{\iota(j,k)}} \tau_{j,k,m}(t)$$

$\omega_{j,k}(t)$ ≡ (omega) dual variable that is complementarily slack with $\dot{u}_{j,k}(t)$ and $\Delta u_{j,k}(t)$ (marginal price of flow conservation for jobs of type j in stage k at time t)

$\boldsymbol{\omega}(t)$ ≡ vector of dual variables that are complementarily slack with $\dot{\mathbf{u}}(t)$ and $\Delta\mathbf{u}(t)$ where element number $\kappa(j,k)$ has the value $\omega_{j,k}(t)$

$\boldsymbol{\Omega}$ ≡ (omega) matrix of dual variables that are complementarily slack with $\dot{\mathbf{U}}$ and $\Delta\mathbf{U}$

$= \left[\boldsymbol{\omega}(t_1) \quad \boldsymbol{\omega}(t_2) \quad \cdots \quad \boldsymbol{\omega}(t_N)\right]$

C.11 LATIN SYMBOLS

C.11.1 Data Variables

a_i ≡ expected fraction of time that each machine of type i is available for processing (so the average number of available machines of type i over time is $a_i M_i$)

a ≡ vector of machine availabilities where element number i has the value a_i, so the average number of available machines over time is $\mathbf{D(a)m}$

A ≡ constraint matrix in QP and LP formulations

b ≡ vector of constraint right-hand sides in QP and LP formulations

B ≡ binary constituency matrix of zeros and ones showing the assignment of machines to job stages

$= \left\{ B_{i,k} : \begin{matrix} i = 1,2,\ldots,I \\ k = 1,2,\ldots,K \end{matrix} \right\}$ where $B_{i,k} \equiv \begin{cases} 1 & \text{if } \exists (j,k') \in \sigma_i : \kappa(j,k') = k \\ 0 & \text{otherwise} \end{cases}$

$C(m,k)$ ≡ number of combinations of m indistinguishable items taken k at a time

$$= \binom{M+K-1}{K}$$

$$= \frac{(M+K-1)!}{K!(M-1)!}$$

c ≡ constant in objective function of LP formulation

$c_{j,k}$ ≡ holding cost per unit time for jobs of type j in stage k

$c^{+}_{j,k}$ ≡ keeping cost (holding cost per unit time for jobs of type j in stage k minus holding cost in stage $k + 1$, which is zero if $k = k_j$)

$$= c_{j,k} - c_{j,k+1}$$

$\mathbf{c}$ ≡ vector of holding costs per unit time where element number $\kappa(j,k)$ has the value $c_{j,k}$

$\mathbf{c}^{+}$ ≡ vector of keeping costs per unit time where element number $\kappa(j,k)$ has the value $c^{+}_{j,k}$

$$= \left(\mathbf{I}_K - \mathbf{P}\right)\mathbf{c}$$

$d_{j,l}$ ≡ due date for lth job of type j

$\mathbf{D}(\cdot)$ ≡ diagonal matrix formed from any vector where diagonal element number i in $\mathbf{D}(\mathbf{x})$ has the value x_i

$\mathbf{e}_n$ ≡ unit vector of all zeros except a one in the nth position

$\mathbf{f}$ ≡ vector of constants for linear term in objective function of QP and LP formulations

$\mathbf{H}$ ≡ symmetric Hessian matrix of constants for quadratic term in objective function of QP formulation

$\mathbf{I}$ ≡ identity matrix with dimensions given by context (such as $\mathbf{I} - \mathbf{R}$) or by subscript (such as $\mathbf{I}_K$)

$= \mathbf{D}(\mathbf{1})$

$$= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

I ≡ number of machine types

J ≡ number of job types

K_j ≡ number of stages for jobs of type j

$K_{\max}$ ≡ largest number of stages for any job

$= \max\left\{K_j : j = 1, 2, \ldots, J\right\}$

K ≡ total number of stages for all jobs

$= \sum_{j=1}^{J} K_j$

L ≡ unit lower triangular matrix of ones (with size given by context)

$$= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

M_i ≡ number of machines of type i

$\hat{M}_i$ ≡ actual number of machines of type i that are available (idle or processing, but not broken down) at time t_0

m ≡ vector of numbers of available machines of each type where element number i has the value M_i

m̂ ≡ vector of actual number of machines that are available at time t_0 where element number i has the value $\hat{M}_i$

$M_{\max}$ ≡ number of machines in most plentiful machine type

$$= \max\left\{M_i : i = 1, 2, \ldots, I\right\}$$

M ≡ total number of machines of all types

$$= \sum_{i=1}^{I} M_i$$

N ≡ number of time intervals

$p_{j,k}$ ≡ mean processing time for jobs of type j in stage k (or the inverse of the average processing rate)

$$= \frac{1}{\mu_{j,k}}$$

p_j ≡ expected total length of time to completely process a job of type j

$$= \sum_{k=1}^{K_j} p_{j,k}$$

$p_{\max}$ ≡ largest expected total length of time to completely process any job

$$= \max\left\{p_j : j = 1, 2, \ldots, J\right\}$$

p ≡ expected total length of time to completely process one of each type of job

$$= \sum_{j=1}^{J} p_j$$

p ≡ vector of mean processing times where element number $\kappa(j,k)$ has the value $p_{j,k}$

$\mathbf{P}$ ≡ job process routing (from-to) matrix which in the *simple* case consists of only zeros with some ones on the first superdiagonal (as shown below), although in more general cases it could have other nonzero elements and elements less than unity if product flows have scrap probabilities or sub-routes (such as rework loops or send-aheads or sample testing) or other probabilistic routings

$$= \begin{cases} \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} & \text{if } J = 1 \\ \text{a block diagonal matrix of such blocks} & \text{if } J > 1 \end{cases}$$

$q_{j,k}(t_0)$ ≡ initial number of jobs of type j in stage k

$q^{+}_{j,k}(t_0)$ ≡ initial total (immediate and upstream) WIP (number of jobs of type j in stage k or earlier)

$$= \sum_{k'=1}^{k} q_{j,k'}(t_0)$$

$\mathbf{q}(t_0)$ ≡ vector of initial number of jobs in queue or in service where element number $\kappa(j,k)$ has the value $q_{j,k}(t_0)$

$\mathbf{q}^{+}(t_0)$ ≡ vector of initial total (immediate and upstream) WIP where element number $\kappa(j,k)$ has the value $q^{+}_{j,k}(t_0)$

$$= \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \mathbf{q}(t_0)$$

q_j ≡ initial number of jobs of type j

$$= \sum_{k=1}^{K_j} q_{j,k}(t_0)$$

$q_{\max}$ ≡ initial number of jobs of most plentiful type

$$= \max\left\{q_j : j = 1,2,\ldots,J\right\}$$

q ≡ total initial number of jobs

$$= \sum_{j=1}^{J} q_j$$

$r_{j,l}$ ≡ release date when the lth job of type j is ready to begin processing

$s_{j,k,j',k'}$ ≡ mean sequence-dependent setup time for a machine to switch from processing jobs of type j in stage k to processing jobs of type j' in stage k'

$\mathbf{S}$ ≡ matrix of mean sequence-dependent setup times where element number $\left[\kappa(j,k),\kappa(j',k')\right]$ has the value $s_{j,k,j',k'}$

t_0 ≡ starting time (usually assumed to be zero)

T ≡ time horizon over which to optimize (or time horizon over which jobs arrive or time horizon after which arrivals can be ignored)

T^* ≡ optimal fluid makespan objective

$$= \max\left\{\frac{\sum_{(j,k)\in\sigma_i} p_{j,k} q^+_{j,k}(t_0)}{a_i M_i - \sum_{(j,k)\in\sigma_i} p_{j,k}\alpha^+_{j,k}} : i = 1,2,\ldots,I\right\}$$

$$= \left\| \left\{ \mathbf{D}\left[\mathbf{D}(\mathbf{a})\mathbf{m} - \mathbf{B}\mathbf{D}(\mathbf{p})\boldsymbol{\alpha}^+ \right] \right\}^{-1} \mathbf{B}\mathbf{D}(\mathbf{p})\mathbf{q}^+(t_0) \right\|_\infty$$

$u_{j,k,m}(t_0)$ ≡ initial number of units of job type j that have finished processing on machine m at stage k

$u_{j,k}(t_0)$ ≡ initial number of units of job type j that have finished processing at stage k

$$= \sum_{m=1}^{M_{\iota(j,k)}} u_{j,k,m}(t_0)$$

$\mathbf{u}(t_0)$ ≡ vector of initial number of jobs that have finished processing

$w_i(t_0)$ ≡ initial immediate workload (time to process all jobs initially waiting) at machine i

$$= \frac{\sum_{(j,k)\in\sigma_i} p_{j,k} q_{j,k}(t_0)}{a_i M_i}$$

$w_{\max}(t_0)$ ≡ maximum initial immediate workload over all machines

$$= \max\left\{w_i(t_0) : i = 1,2,\ldots,I\right\}$$

$$= \left\|\mathbf{w}(t_0)\right\|_\infty$$

$\mathbf{w}(t_0)$ ≡ vector of initial immediate machine workloads

$$= \left[\mathbf{D(a)D(m)}\right]^{-1}\mathbf{BD(p)q}(t_0)$$

C.11.2 Decision Variables

C_{h} ≡ holding cost objective function (total weighted holding cost of solution schedule)

$C_{j,k,l}$ ≡ completion time of lth job of type j in stage k

$$= \min\left\{t : u_{j,k}(t) \geq l\right\}$$

$C_{\max}$ ≡ makespan objective function (time to complete all jobs in solution schedule)

$$= \max\left\{C_{j,K_j,l} : \begin{matrix} j = 1,2,\ldots,J \\ l = 1,2,\ldots \end{matrix}\right\}$$

C_{sum} ≡ objective function that is a weighted sum of C_{h} and C_{w}

C_{w} ≡ maximum workload objective function

$E_{j,l}$ ≡ earliness of lth job of type j

$$= \max\left\{0, d_{j,l} - C_{j,K_j,l}\right\}$$

$$= \max\left\{0, L_{j,l}\right\}$$

$L_{j,l}$ ≡ lateness of lth job of type j

$$= C_{j,K_j,l} - d_{j,l}$$

$L_{\max}$ ≡ maximum lateness

$$= \max\left\{L_{j,l} : \begin{matrix} j = 1,2,\ldots,J \\ l = 1,2,\ldots \end{matrix}\right\}$$

$q_{j,k}(t)$ ≡ number of jobs of type j in queue or in service at stage k at time t

$q^{+}_{j,k}(t)$ ≡ total (immediate and upstream) WIP (number of jobs of type j in stage k or earlier)

$$= \sum_{k'=1}^{k} q_{j,k'}(t)$$

$\mathbf{q}(t)$ ≡ vector of number of jobs in queue or in service where element number $\kappa(j,k)$ has the value $q_{j,k}(t)$

$\mathbf{q}^{+}(t)$ ≡ vector of total (immediate and upstream) WIP where element number $\kappa(j,k)$ has the value $q^{+}_{j,k}(t)$

$= \left(\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}\right)^{-1} \mathbf{q}(t)$

$\mathbf{Q}$ ≡ matrix of number of jobs in queue or in service

$= \left[\mathbf{q}(t_1) \quad \mathbf{q}(t_2) \quad \cdots \quad \mathbf{q}(t_{N-1})\right]$

$\mathbf{Q}^{+}$ ≡ matrix of total (immediate and upstream) WIP

$= \left[\mathbf{q}^{+}(t_1) \quad \mathbf{q}^{+}(t_2) \quad \cdots \quad \mathbf{q}^{+}(t_{N-1})\right]$

$\Delta q_{j,k}(t_n)$ ≡ increase in number of jobs of type j in queue or in service at stage k in the nth time interval

$= q_{j,k}(t_n) - q_{j,k}(t_{n-1})$

$\Delta q^{+}_{j,k}(t_n)$ ≡ increase in total (immediate and upstream) WIP of jobs of type j in queue or in service at stage k or earlier in the nth time interval

$= q^{+}_{j,k}(t_n) - q^{+}_{j,k}(t_{n-1})$

$\Delta\mathbf{q}(t_n)$ ≡ vector of increase in number of jobs in queue or in service in the nth time interval

$= \mathbf{q}(t_n) - \mathbf{q}(t_{n-1})$

$\Delta\mathbf{q}^{+}(t_n)$ ≡ vector of increases in total (immediate and upstream) WIP in the nth time interval

$= \mathbf{q}^{+}(t_n) - \mathbf{q}^{+}(t_{n-1})$

$\Delta\mathbf{Q}$ ≡ matrix of increase in number of jobs in queue or in service

$= \left[\Delta\mathbf{q}(t_1) \quad \Delta\mathbf{q}(t_2) \quad \cdots \quad \Delta\mathbf{q}(t_{N-1})\right]$

$\Delta\mathbf{Q}^{+}$ ≡ matrix of increases in total (immediate and upstream) WIP

$= \left[\Delta\mathbf{q}^{+}(t_1) \quad \Delta\mathbf{q}^{+}(t_2) \quad \cdots \quad \Delta\mathbf{q}^{+}(t_{N-1})\right]$

$T_{j,l}$ ≡ tardiness of lth job of type j

$= \max\left\{0, C_{j,K_j,l} - d_{j,l}\right\}$

$= \max\left\{0, L_{j,l}\right\}$

t_n ≡ end of nth time interval

Δt_n ≡ length of nth time interval

$= t_n - t_{n-1}$

$\Delta\mathbf{t}$ ≡ vector of time interval lengths where element number n has the value Δt_n

$= \left[\Delta t_1 \quad \Delta t_2 \quad \cdots \quad \Delta t_N\right]^{\mathrm{T}}$

$u_{j,k,m}(t)$ ≡ cumulative number of units of job type j that have finished processing on machine m at stage k by time t

$u_{j,k}(t)$ ≡ cumulative number of units of job type j that have finished processing at stage k by time t

$= \sum_{m=1}^{M_{\iota(j,k)}} u_{j,k,m}(t)$

$= \max\left\{l : C_{j,k,l} \le t\right\}$

$\mathbf{u}(t)$ ≡ vector of cumulative number of units that have finished processing where element number $\kappa(j,k)$ has the value $u_{j,k}(t)$

$\Delta u_{j,k}(t_n)$ ≡ number of units of job type j that finish processing at stage k in the nth time interval

$= u_{j,k}(t_n) - u_{j,k}(t_{n-1})$

$\Delta \mathbf{u}(t_n)$ ≡ vector of number of units that finish processing in nth time interval

$= \mathbf{u}(t_n) - \mathbf{u}(t_{n-1})$

$\Delta \mathbf{U}$ ≡ matrix of number of units that finish processing

$= \left[\Delta \mathbf{u}(t_1) \quad \Delta \mathbf{u}(t_2) \quad \cdots \quad \Delta \mathbf{u}(t_N)\right]$

$v_{j,k}(t)$ ≡ equivalent (possibly non-integer) number of machines working on jobs of type j in stage k at time t

$= p_{j,k}\dot{u}_{j,k}(t)$

$\mathbf{v}(t)$ ≡ vector of equivalent number of machines working

$= \mathbf{D}(\mathbf{p})\dot{\mathbf{u}}(t)$

$\breve{\mathbf{v}}$ ≡ deviation from integrality in the number of machines working

$= \left\| \hat{\mathbf{v}}(t_0^+) - \tilde{\mathbf{v}}^*(t_1) \right\|$

$w_i(t)$ ≡ immediate workload (time to process all jobs currently waiting) at machine i

$$= \frac{\sum_{(j,k)\in\sigma_i} p_{j,k} q_{j,k}(t)}{a_i M_i}$$

$w_{\max}(t)$ ≡ maximum immediate workload over all machines

$= \max\left\{ w_i(t) : i = 1, 2, \ldots, I \right\}$

$= \left\| \mathbf{w}(t) \right\|_\infty$

$\mathbf{w}(t)$ ≡ vector of immediate machine workloads

$= \left[\mathbf{D}(\mathbf{a})\mathbf{D}(\mathbf{m})\right]^{-1} \mathbf{B}\mathbf{D}(\mathbf{p})\mathbf{q}(t)$

$\mathbf{x}$ ≡ vector of decision variables in QP and LP formulations

$y_i(t)$ ≡ cumulative unused capacity lost on machines of type i in the time interval $\left[t_0 - t\right]$ (so $\dot{y}_i(t)$ is the possibly non-integer equivalent number of machines of type i that are idle at time t)

$\mathbf{y}(t)$ ≡ vector of unused capacity lost where element number i has the value $y_i(t)$

$\Delta\mathbf{y}(t_n)$ ≡ vector of unused capacity lost in nth time interval

$= \mathbf{y}(t_n) - \mathbf{y}(t_{n-1})$

$\Delta\mathbf{Y}$ ≡ matrix of unused capacity lost

$= \left[\Delta\mathbf{y}(t_1) \quad \Delta\mathbf{y}(t_2) \quad \cdots \quad \Delta\mathbf{y}(t_N)\right]$

$\mathbf{z}$ ≡ vector of dual decision variables in DQP and DLP formulations

C.12 NOTATION COMPARISON

The following tables compare the notation used in this document (which follows Pinedo [2002] as much as possible) with that of Weiss [2001] and that of Bertsimas & Sethuraman [1999] and Bertsimas, Gamarnik & Sethuraman [2000] where the notation differs for common concepts.

Table C.1: Comparison of This Document's Notation with That of Weiss

Discrete Indices		Data Variables		Decision Variables	
Weiss	Billings	Weiss	Billings	Weiss	Billings
i	j	$\mathbf{a}$	$\mathbf{q}(t_0)$	$\boldsymbol{\theta}(t)$	$\dot{\boldsymbol{\omega}}(t)$
j	k	$\mathbf{b}$	$\mathbf{D(a)m}$	$\boldsymbol{\xi}(t)$	$\dot{\mathbf{q}}(t)$
k	k	$\mathbf{c}$	$\mathbf{c}^+$	τ_n	Δt_n
l	k, n	C_i	σ_i	$Q_k(t)$	$q_{j,k}(t)$
n	l	$\mathbf{G}$	$\mathbf{I}_K - \mathbf{P}^{\mathrm{T}}$	$\mathbf{q}(t)$	$\boldsymbol{\omega}(t)$
r	j, n	$\mathbf{H}$	$\mathbf{BD(p)}$	$\mathbf{r}(t)$	$\boldsymbol{\xi}(t)$
		I	I	R	N
		J	K	$T^{(r)}$	t_n
		K	K	$\mathbf{u}(t)$	$\dot{\mathbf{u}}(t)$
		K_r	K_j	$\mathbf{x}(t)$	$\mathbf{q}(t)$
		$\mathbf{M}$	$\mathbf{BD(p)}$	$\mathbf{z}(t)$	$\mathbf{y}(t)$
		m_k	$p_{j,k}$		
		$\mathbf{w}$	$\mathbf{c}$		

Table C.2: Comparison of This Document's Notation With That of Bertsimas et al.

Discrete Indices		Data Variables		Decision Variables	
Bertsimas	Billings	Bertsimas	Billings	Bertsimas	Billings
i	j	α_i	$q_{j,1}(t_0)/q$	$DC_{i,k}(n)$	$\hat{C}_{j,k,l}$
j	i	σ_i^k	$\iota(j,k)$	$FC_{i,k}(n)$	$\tilde{C}_{j,k,l}$
k	k	C_j	$t\theta_i(t)$	L^r	Δt_n
l	k	$C_{\max}$	$t\theta_{\max}(t)$	$m_{i,k}(t)$	$u_{j,k}(t)$
n	l	I	J	$n_{i,k}(t)$	$q_{j,k}(t)$
N	q	J	I	R	N
r	n	J_i	K_j	T^r	t_n
		$J_{\max}$	$K_{\max}$	$T_{i,k}(t)$	$\tau_{j,k}(t)$
		L	p	$u_{i,k}(t)$	$\dot{\tau}_{j,k}(t)$
		L_i	p_j	$v_{i,k}$	$v_{j,k}(t)$
		$L_{\max}$	$p_{\max}$	$x_{i,k}(t)$	$\tilde{q}_{j,k}(t)$
		n_i	$q_{j,1}(t_0)$	$y_{i,k}^r$	$\Delta u_{j,k}(t_n)$
		P_j	ψ_i		
		$P_{\max}$	$\psi_{\max}$		
		U_j	ϕ_i		
		$U_{\max}$	$\phi_{\max}$		
		$w_{i,k}$	$c_{j,k}$		
		x_i	$q_{j,1}(t_0)$		

Bibliography

Anderson, E. J. [1981]. A new continuous model for job-shop scheduling. *International J. Systems Science* 12, 1469-1475.

Anderson, E. J. and P. Nash. [1987]. *Linear Programming in Infinite Dimensional Spaces*. Wiley- Interscience, Chichester.

Avram, F., D. Bertsimas, and M. Ricard. [1995]. "Fluid models of sequencing problems in open queueing networks: an optimal control approach." In *Stochastic Networks*, volume 71 of IMA Volumes in Mathematics and its Applications, F.P. Kelly and R. Williams, eds. Springer, New York, 199-234

Bertsimas, D., I. Paschalidis and J. Tsitsiklis. [1994] "Optimization of multiclass queueing networks: polyhedral and nonlinear characterizations of achievable performance", (with), *Annals of Applied Probability*, 4, 1, 43-75, 1994.

Bertsimas, D., Gamarnik, D. [1999]. Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2): pp.296-318, 1999

Bertsimas, D., D. Gamarnik, and J. Sethuraman, [1999]. "From fluid relaxations to practical algorithms for job shop scheduling: the holding cost objective", *Operations Research* (to appear).

Bertsimas, D. and J. Sethuraman, [2002]. "From fluid relaxations to practical algorithms for job shop scheduling: the makespan objective", *Math Programming*, Series A, 2002.

Boudoukh, T., Penn, M., Weiss, G. [1998] Scheduling jobshops with some identical or similar jobs. Preprint .

Campbell, E. and J. Ammenheuser, [1999] *300-mm Factory Layout and Material Handling Modeling: Phase II Report*, Tech Transfer Document #99113848BENG, International SeMaTech, November, 1999

Chen, R-R and S.P. Meyn, [1999] Value Iteration and Optimization of Multiclass Queueing Networks , *Queueing Systems*, 32, pp. 65--97, .

Chen, H. and D.D. Yao, [1993] "Dynamic scheduling of a multi-class fluid network", *Operations Research*, 41, 1104-1115 (1993).

Coffman, E.G. and Lueker, G.S. [1991] *Probabilistic Analysis of Packing and Partitioning Algorithms* Wiley, New York.

Cox, D.R. and W.L. Smith [1961]. *Queues*. Methuen, London.

Dai, J.G. [1995]. On positive Harris recurrence of multi-class queueing networks: a unified approach via fluid limit models. *Annals of Applied Probability* 5, 49-77.

Dai, J. and Weiss, G. [1996] "Stability and Instability of Fluid Models for certain Re-Entrant Lines" *Mathematics of Operations Research* 21, pp. 115-134, .

Dai, J. G. and G. Weiss. [2002]. "A Fluid Heuristic for Minimizing Makespan in Job-Shops." *Operations Research* 50, pp. 692-707.

Feigin, G., J. Fowler, J. Robinson, and R. Leachman. [1994] *Semiconductor Wafer Manufacturing Data Format Specification* International SeMaTech, ftp://ftp.eas.asu.edu/pub/centers/masmlab/factory_datasets

Fleischer, Lisa and Jay Sethuraman, [2003], Approximately optimal control of fluid networks, pp. 56-65, *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, Maryland

Goldratt, E.M. and Cox, J. [1992] *The Goal: A Process of Ongoing Improvement*. 2nd edition North River Press, .

Harrison, J.M., [1996]. The BIGSTEP approach to flow management in stochastic processing networks, *Stochastic Networks: Theory and Applications*, F. P. Kelly, I. Ziedins, S. Zachary, Editors, Oxford University Press.

Harrison, J.M., [1988]. Brownian models of queueing networks with heterogeneous customer populations. *Proceedings of the IMA Workshop on Stochastic Differential Systems*, Fleming W., Lions P.L., editors, Springer-Verlag.

Harrison, J.M., and L. M. Wein [1989] Scheduling Networks of Queues: Heavy Traffic Analysis of a Simple Open Network, *Queueing Systems* 5, 265-280

Klimov, G. P. [1974] Time-sharing service systems. *J-Theory-Probab-Appl,* volume 19

Kushner, H.J. and Ramachandran, K.M. [1989] Optimal and approximately optimal control policies for queues in heavy traffic. *SIAM J Control and Optimization* 27, 1293-1318.

Little, J. D. C. [1961]. "A Proof for the Queueing Formula $L = \lambda W$." *Operations Research* **9**, 383-397.

Luo, X. and D. Bertsimas. [1998]. "A new algorithm for state constrained separated continuous linear programs." To appear in *SIAM J. Control and Optimization*.

Maglaras, Costis [2000] Discrete-review policies for scheduling stochastic networks: Trajectory tracking and fluid-scale asymptotic optimality. *Annals of Applied Probability*, 10(3): 897-929, .

Maglaras, Costis [2003] Continuous-review tracking policies for dynamic control of stochastic networks, *QUESTA*, 43:43-80, .

Martins, L. F., S. Shreve, and H. M. Soner [1996] Heavy traffic convergence of a controlled, multi-class queueing system, *SIAM Journal on Control and Optimization* 34 (1996), 2133-2171

Meyn, S.P [1995] The policy improvement algorithm for Markov decision processes with general state space. *IEEE Transactions on Automatic Control*, to appear.

Meyn, S.P [2001] "Sequencing and Routing in Multiclass Queueing Networks I: Feedback Regulation," *SIAM Journal on Control and Optimization*, 2001.

Meyn, S.P [2003] Sequencing and Routing in Multiclass Queueing Networks. Part II: Workload Relaxations, *SIAM J. Control and Optimization*, Vol. 42, No. 1, pp. 178-217, 2003.

Neuts, Marcel F. [1981] *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*, Baltimore: Johns Hopkins University Press,

Pullan, Malcolm C. [1993]. An algorithm for a class of continuous linear programs, *SIAM Journal on Control and Optimization*, Volume 31 Issue 6, November 1993

Pullan, Malcolm C. [1995]. Forms of optimal solutions for separated continuous linear programs. *SIAM Journal on Control and Optimization*, Volume 33 Issue 6, pp. 1952–1977. November 1995

Pullan, Malcolm C. [1996]. A Duality Theory for Separated Continuous Linear Programs, *SIAM Journal on Control and Optimization*, Volume 34 Issue 3, May 1996

Pullan, Malcolm C. [1997]. A Study of General Dynamic Network Programs with Arc Time-Delays, *SIAM Journal on Optimization,* Volume 7 Issue 4, April 1997

Pullan, Malcolm C. [1999]. Convergence of a General Class of Algorithms for Separated Continuous Linear Programs, *SIAM Journal on Optimization*, Volume 10, Issue 3, July 1999

Quinn, T., and E. Bass [1999] *300 mm Factory Layout and Material Handling Modeling: Phase I Report*, Technology Transfer #99023688B-ENG, http://www.sematech.org/public/docubase/document/3688beng.pdf International SeMaTech December, 1999,

Sahinidis, N. V., [1996] BARON: A General Purpose Global Optimization Software Package, *Journal of Global Optimization*, 8(2), 201-205, .

Sevastyanov, S.V. [1994] On some geometric methods in scheduling theory, a survey. *Discrete Applied Mathematics* 55, 59-82.

Schruben, L. W. [1997] *Graphical Simulation Modeling and Analysis: Using Sigma for Windows* Boyd & Fraser Pub Co; (July 1997)

Serfozo, Richard [1999] *Introduction to Stochastic Networks*. Springer-Verlag, New York,

Stanley, T., R. Wright, M. Brown, K. Rust, B. van Eck, J. Pace, A. Ghatalia, R. Billings, T. Breeden, K. Torres, G. Williamson, J. Barnett, and J. W. Bailey, [2001] *Q2 2001 Productivity Analysis Factory & Cost Simulation Experiment Report*, International SeMaTech, 2001 http://www.sematech.org/public/resources/simulation/fab.htm,

Vandergraft, James [1983] Fluid Flow Model of Networks of Queues *Management Science*

Veatch, M. H., [2001], Fluid analysis of arrival routing. *IEEE Transactions on Automatic Control* 46: 1254-1257

Veatch, M. H., [2002], Using fluid solutions in dynamic scheduling. In S. B. Gershwin, Y. Dallery, C. T. Papadopoulos, J. M. Smith, eds., *Analysis and modeling of manufacturing systems*, pp. 399-426. Kluwer, New York.

Wein, L.M., [1992]. Scheduling networks of queues: heavy traffic analysis of a multistation network with controllable inputs. *Operations Research* 40, S312--S334.

Weiss, G. [1995] On optimal draining of fluid re-entrant lines. In *Stochastic Networks - IMA Volumes in Mathematics and its Applications*, Editors: F.P. Kelly and Ruth Williams, Springer-Verlag, New York, 93-105.

Weiss, G. [2002] "A Simplex Based Algorithm to Solve Separated Continuous Linear Programs" Submitted for publication to *Mathematical Programming*, 2002

Yaffe H. [1974] Model for Optimal Operation and Design of Solid Waste Transfer Stations, *Transportation Science*

Yang, Bingyi, [1996], *Queueing networks with string transitions and shared resources*, Thesis (Ph.D.), School of Industrial and Systems Engineering, Georgia Institute of Technology Directed by Richard F. Serfozo.

Vita

Ronald Lester Billings was born on January 23, 1958, in Everett, Washington. His parents were Ruth Louise Allen Billings (born in Dali, Yunnan, China) and David Elliott Billings (born in Lebanon, Oregon). Dr. Billings grew up in the Philippines, and he finished high school in Portland, Oregon in 1976. He served in the U.S. Army from 1977 to 1980 as a military bandsman at Fort Hood, Texas, and worked for the U.S. Air Force from 1980 to 1982 as a civilian radiographic file technician in San Antonio, Texas. From 1983 to 1990, Dr. Billings worked for the University of Texas at Austin as a Resident Assistant, Graduate Teaching Assistant, Continuing Engineering Instructor, and Graduate Research Assistant.

From 1991 to 2002, Dr. Billings worked at SEMATECH (a consortium of semiconductor manufacturing companies) in Austin, Texas, as an Engineering Editor/Team Leader, Automation Engineer, Factory Architecture Group Manager, and Material Logistics Standards Project Manager. From 2002 to the present, Dr. Billings has served as Partner and CEO for Fluid Analysis for Balancing Queues (FABQ.com) in Austin, Texas, developing software for factory scheduling and dispatching using fluid models.

Dr. Billings' higher education includes a B.S. in Electrical Engineering with Honors (1986), an M.B.A. (1991), an M.S. in (Mechanical) Engineering (1993), an M.S. in Computational and Applied Mathematics (1996), and this PhD. in Operations Research and Industrial Engineering (all from the University of Texas at Austin).

In the Spring of 2003, Dr. Billings taught Quantitative Literacy (Math for Liberal Arts Students) at Concordia University in Austin, Texas. In August of 2003, Dr. Billings accepted a position as Assistant Professor at Georgia Institute of Technology's School of Industrial and Systems Engineering in Atlanta, GA.

Permanent address: School of Industrial and Systems Engineering
Georgia Institute of Technology
765 Ferst Drive
Atlanta, GA 30332-0205

This dissertation was typed by the author.